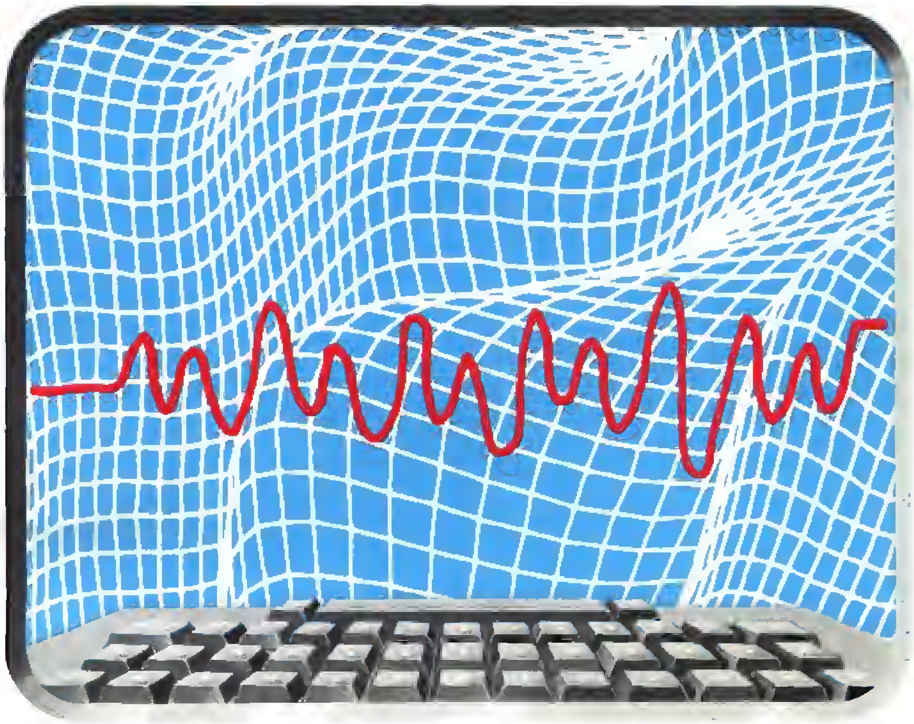
NO. 60 MAY 1983 U.S. Edition: $2.50 International Edition: $3.00

MICRO™

Advancing Computer Knowledge

New Wave of Computers

- ATARI 1200XL
- APPLE IIe
- C64...



Color Computer Clock

Graphics on the APPLE and PET

In this month's Learning Center:

File Management for Commodore Computers

Breakup: An Animation Game for APPLE and Commodore...

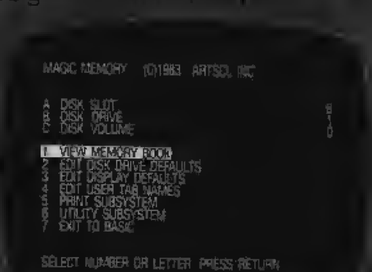
MAGIC MEMORY

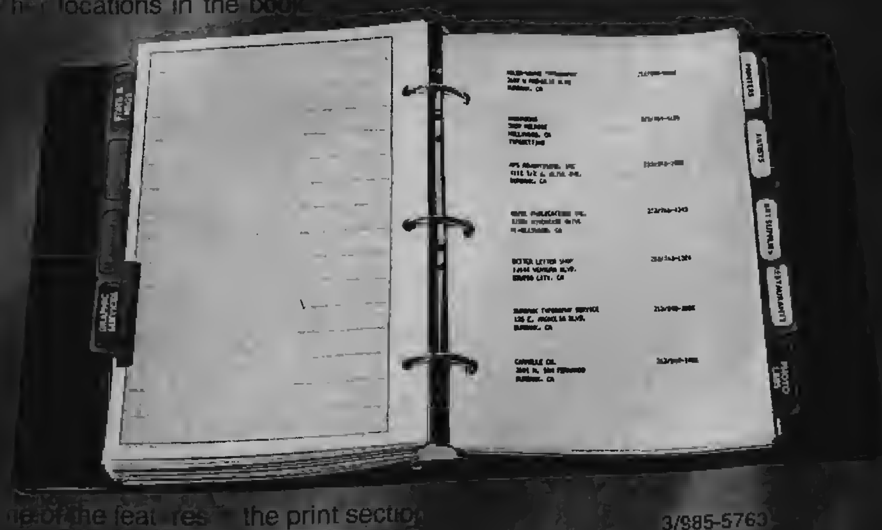
bus ness cards etc

...ersatile than the ... This system can ... delete informa... alphabetically, and transfe... locations in the book.

...proudly announces ... the complete advanced database system MAGIC MEMORY ...

... as been designed ...





...of the features ... the print section ... MAGIC MEMORY is the ability ... the *address* ...

3/985-5763
5547 S... Avenue
H..., Ca 9160

MICRO™ Highlights

The New Wave of Computers

"Great waves looked over others coming in...." Robert Frost was speaking of the Pacific Ocean; we are speaking of the many new computer products that are flooding the market. A few years ago it was a simple matter to go out and buy a home computer — you didn't have much selection. Today you must choose from computers that fit nearly any situation, any pocketbook, and any kind of computerist — from novice to highly technical.

Once you have decided which computer is right for you (taking into consideration memory requirements, load and save capabilities, editing functions, etc.), then you should become knowledgeable about add-on boards, modems, printers, hardware, and software that is available. The May issue of MICRO is designed to provide you with information about the tide of new products flowing into the market today and what you can expect in the near future.

Paul Swanson opens our feature section with a comparison of the Atari 1200XL, 400, and 800 in "The Atari 1200XL" (pg. 20). In "Microprocessors for Your Apple II" (pg. 26), Phil Daley and John Hedderman compare add-on boards for the Apple II that increase capabilities, provide new features, and teach new languages. "New Commodores" (pg. 30) by Loren Wright is a discussion of Commodore's new product line, including the VIC-20, Commodore 64, and a computer soon to appear on the market — the C 128/80. Keith Roberson asks "Is 16 Bits the Solution?" (pg. 32). The 16-bit microprocessor has made a significant impact on the imagination of microprocessor users. However, Keith believes 8-bit technology can be expected to have a long and active life in the small computer marketplace.

Programming Aids

This month we provide you with seven programming aids. Werner Kolbe shows you how to write up to four times more numerical data on a CBM disk using the computer's internal binary format instead of ASCII strings. Read "Get More Data on Your CBM Disk" (pg. 38). Randall Hyde discusses several methods for passing data to and from assembly-language subroutines. He uses examples for the 6502, 6809, 68000, and 16032 microprocessors. See "Parameter Passing in Assembly Language" (pg. 40). John Steiner has written a subroutine that adds a real-time clock and date function to your programs. Learn how to use TIME$/SUB in "Real-time Clock for the Color Computer" (pg. 50).

"&GET" by David Dice (pg. 52) is a machine-language program to allow customized input routines; and "Modifying and Using MAE" (pg. 56), by F. Arthur Cochrane, describes how to modify the MAE assembler to enable output to an ASCII printer and listing output to a disk file. Joe Hootman continues his series with a discussion of miscellaneous instructions implemented by the 68000. See "68000 Instructions" (pg. 58). And finally, Jim Strasma continues his series "It's All Relative, Part 5: Printing Commodore File Data" (pg. 62) with a discussion on formatting and printing data from Commodore relative disk files.

Graphics and Printers

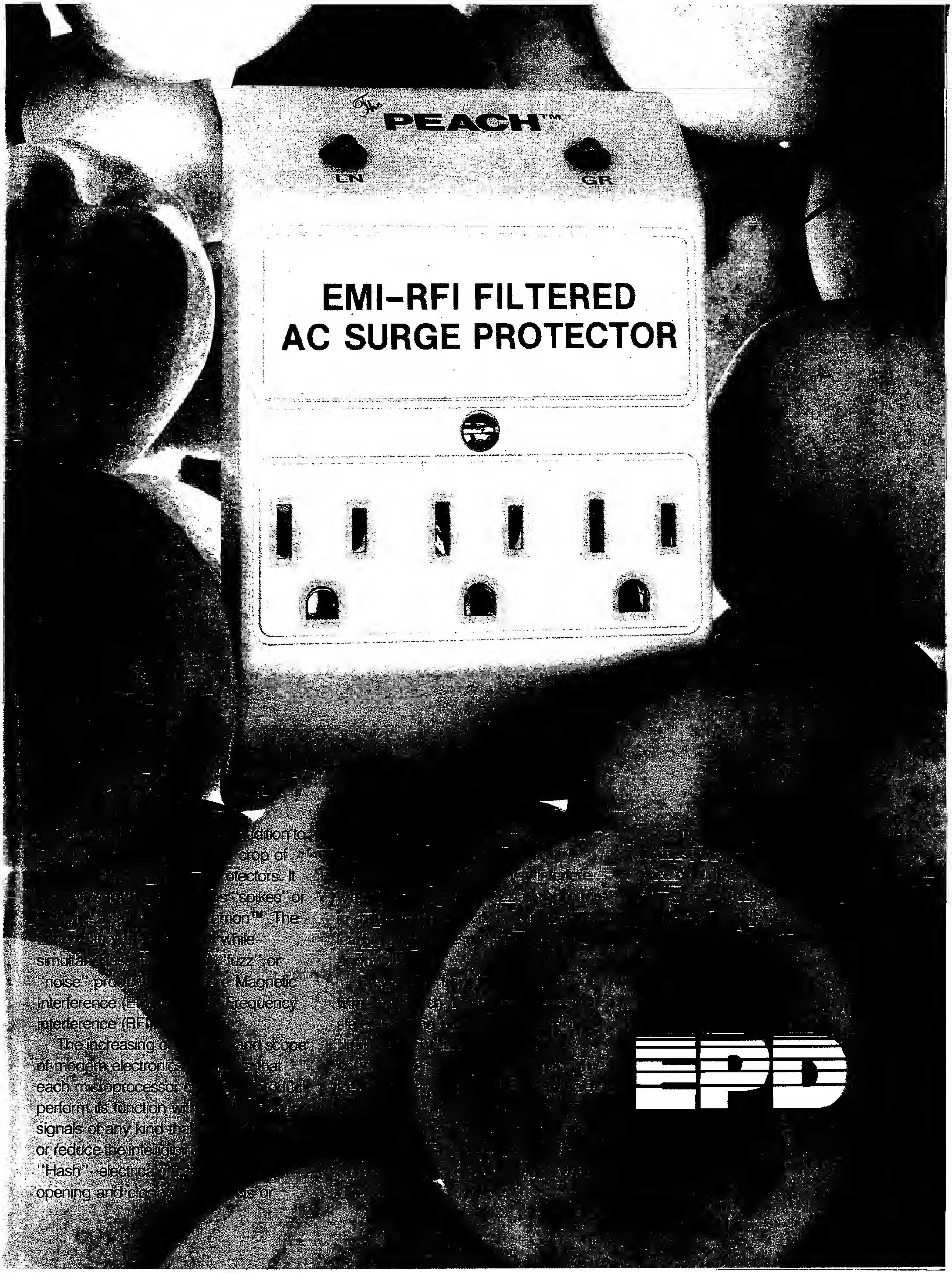
The graphics section includes several informative articles regarding printers. You can learn how to print Apple II's hi-res graphics screens on Okidata Microline 80's printer; use an assembly-language program for the Apple II that automatically produces top and bottom margins on each page of printer output, address a second PET as an IEEE device 4. Also learn to print AIM listings the full width of your printer with a machine-language printer driver, and to interface an OSI Superboard II to a Radio Shack Quick Printer II.

The Learning Center

In the Learning Center this month you will study PEEKs and POKEs in "Breakup" (pg. 71) by Loren Wright and Phil Daley. "Breakup" is a simple animation display game that includes a ball and brick wall and tests for collisions. Brian Zupke's "Addressfile" (pg. 76) is an easy tape-based file-management system for the VIC-20 and other Commodore computers. You can store names, addresses, phone numbers, and memos in a cassette file.

We continue to provide you with a wealth of information in our on-going columns. Be sure to read PET Vet, From Here to Atari, Coco Bits, and Interface Clinic, as well as our regular departments. MICRO keeps you on the crest of the wave of new computers.

MICRO


The PEACH™
LN
GR
EMI-RFI FILTERED
AC SURGE PROTECTOR
EPD

MICRO™

Advancing Computer Knowledge

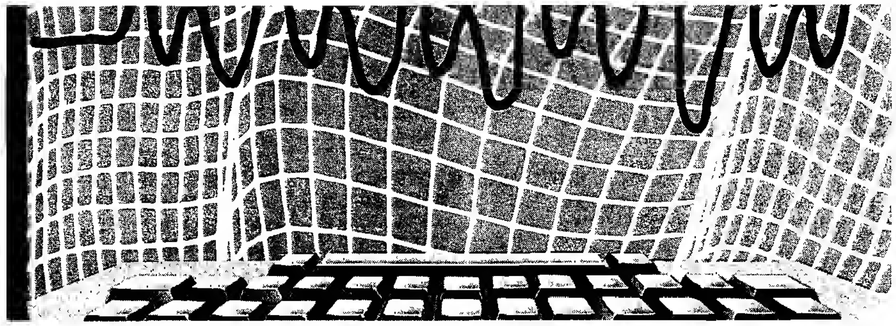
MICRO
P.O. Box 6502
Amherst, NH 03031
(603) 889-4330

MICRO is published monthly by: MICRO, Amherst, NH 03031. Second Class postage paid at: Amherst, NH 03031 and additional mailing offices. USPS Publication Number: 483470. ISSN: 0271-9002. Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to MICRO, 10 Northern Blvd., P.O. Box 6502, Amherst, NH 03031, or call (603) 889-4330, Telex: 955329 TLX SRVC, 800-227-1617. *Subscription rates (per year):* U.S. $24.00, $42.00 / 2 yr. Foreign surface mail $27.00. Air mail: Europe $42.00; Mexico, Central America, Middle East, North Africa, Central Africa $48.00; South America, South Africa, Far East, Australasia, New Zealand $72.00.

New Wave of Computers

PROGRAMMING AIDS

New computers change the nature of the microcomputer industry.



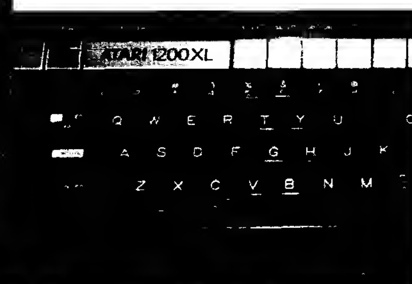
THE LEARNING CENTER

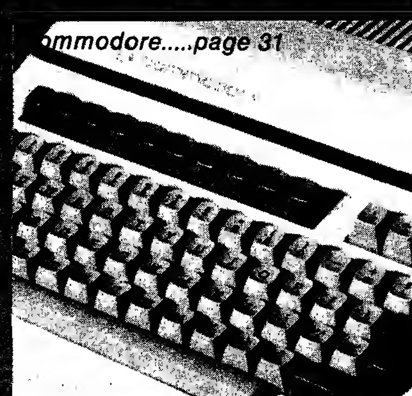
GRAPHICS

COLUMNS

DEPARTMENTS

Atari 1200XL.....page 21





NEW LOWER PRICES

ASHTON TATE
dBASE II $489.00
BOTTOM LINE STRATEGIST 279.00

C.ITOH
PROWRITER PARALLEL 399.00
F-10 55 . . . . 1799.00
F-10 PARRALLEL/SERIAL 1359.00
GRAPPLER INTERFACE 125.00
PROWRITER II 789.00

CALIFORNIA COMPUTER SYSTEMS
ASYNCHRONOUS INTERFACE $129.00
SYNCHRONOUS INTERFACE 149.00
CALENDAR CLOCK 105.00
RS232 INTERFACE 124.00
PROGRAMMABLE TIMER FOR APPLE...99.00

COMSHARE TARGET MARKETING
PLANNER CALC $79.00
TARGET FINANCIAL MODELING . . . 249.00

CONTINENTAL SOFTWARE
HOME ACCOUNTANT FOR APPLE...$59.00
HOME ACCOUNTANT FOR IBM 109.00

DICTRONICS, INC.
RANDOM HOUSE ELECT. THESAURUS $129.00
PROOF READER 50.00

EAGLE
MONEY DECISIONS $99.00

FORCE II
MATH★ $89.00

FOX & GELLER
dUTIL $68.00
QUICKCODE FOR dBASE II 219.00
QUICKSCREEN 129.00

CALL TOLL-FREE
1-800-523-9511
IN PENNSYLVANIA
1-215-868-8219

T/MAKER COMPANY
T/MAKER III $199.00

VIDEX
ENHANCER II 129.00
VIDEX KEYBOARD ENHANCER 129.00
VIDEX VIDEOTERM FOR APPLE II . . . 269.00

VISICORP
DESKTOP PLAN APPLE II $179.00
DESKTOP PLAN IBM 228.00
VISICALC 179.00
VISICALC ADVANCED VERSION . . . . . CALL
VISICALC BUSINESS FORECASTING . 89.00
VISIFILE APPLE II/IBM . . . . CALL
VISILINK 184.00
VISIPLOT FOR APPLE 159.00
VISITREND/VISIPLOT 228.00

IBM IS A TRADEMARK OF INTERNATIONAL BUSINESS MACHINE, APPLE IS A REGISTERED TRADEMARK OF APPLE COMPUTERS, INC., CROSSTALK IS A TRADEMARK OF MICROSTUFF. INFOSTAR IS A TRADEMARK OF MICROPRO. PRICES, SPECIFICATIONS AND AVAILIBILITY SUBJECT TO CHANGE WITHOUT NOTICE. NOT ALL PROGRAMS AVAILABLE IN ALL FORMATS. PLEASE CALL FOR ADDITIONAL PRODUCT INFORMATION.

MICROHOUSE
P.O. Box 499/1444 Linden Street
Department 302
Bethlehem, PA 18016
Dealers inquiries welcome!

INTEGRAL DATA SYSTEMS
IDS MICROPRISM 480 PRINTER . . . $589.00
IDS PRISM-132 PRINTER 1279.00
IDS PRISM-80 PRINTER 999.00

INTERACTIVE STRUCTURES
PKASSO $135.00

INNOVATIVE SOFTWARE, INC.
TIM III $359.00
GRAPHMAGIC 69.00
MATHEMAGIC 69.00

INFORMATION UNLIMITED SYSTEMS
EASYFILER $349.00
EASYSPELLER 159.00
EASYWRITER II 279.00

MAXELL
FD-1 or FH-1-32 (8" SINGLE SIDED) $39.00
FD-2 (8" DOUBLE SIDED) 49.00
MD-1 or MH-1 (5¼" SINGLE SIDED) 31.00
MD-2 or MH-2 (5¼" DOUBLE SIDED) 48.00

MICROPRO
CALCSTAR $99.00
DATASTAR 165.00
MAILMERGE 149.00
WORDSTAR 279.00
WORDSTAR/MAILMERGE 349.00

MICROSOFT
128K RAM FOR IBM PC $389.00
BASIC COMPILER FOR APPLE II 289.00
128K RAMCARD 389.00
192K RAMCARD 519.00
256K RAMCARD 659.00
64K RAMCARD 259.00
MICROSOFT Z80 PREMIUM PACK . . 489.00
MICROSOFT Z80 SOFTCARD 249.00
MULTIPLAN 199.00
TASC APPLESOFT COMPILER 129.00

NORTH AMERICAN BUSINESS SYSTEMS
THE ANSWER $229.00

PEACHTREE
ACCOUNTS PAYABLE . . . . CALL
ACCOUNTS RECEIVABLE . . . . CALL
GENERAL LEDGER . . . . CALL
INVENTORY . . . . CALL
PAYROLL . . . . CALL
SALES INVOICING . . . . CALL

PERFECT SOFTWARE
PERFECT CALC $139.00
PERFECT FILER 199.00
PERFECT SPELLER 139.00
PERFECT WRITER 199.00

QUADRAM
128K MEMORY EXPANSION $380.00
192K MEMORY EXPANSION 475.00
64K MEMORY EXPANSION 280.00
64K MEMOY UPGRADE 129.00
MICROFAZERS ALL MODELS . . . . CALL
QUADBOARD 64K 459.00
QUADBOARD 128K 599.00
QUADBOARD 192K 699.00
QUADBOARD 256K 779.00

CONTROLLER FOR ELITE I $99.00
RANA ELITE I 299.00
RANA ELITE II 465.00
RANA ELITE III 599.00

SMITH-CORONA
SMITH-CORONA TP-1 . . . . $599.00

SORCIM
SUPERCALC BY SORCIM $219.00
SPELLGUARD 189.00

MICROHOUSE™

YOUR MICRO-COMPUTER PEOPLE

Circle No. 4

Letterbox

Dear Editor:

Mr. Tripp and Mr. Daley did a fine job of outlining two strongly contrasting viewpoints regarding BASIC in the February 1983 dual editorial. I would like to express my complete agreement with both viewpoints, with one reservation.

Mr. Tripp's critique of BASIC is not a critique of BASIC at all. It is a critique of a particular implementation of BASIC, most likely Microsoft BASIC. Every complaint he outlines can be fixed in a properly implemented BASIC, and some of the complaints *have* been fixed in some implementations.

For instance, I took delivery of my personal Wang 2200 in November 1972. Wang 2200 BASIC does indeed make effective use of disk facilities (at first only hard disks were supported because floppy disks had not yet been invented!). Since I also own a CBM 8032/8050 combination and several Apple IIs it pains me considerably when I contemplate the exceptionally poor DOSes these machines have.

The Wang also has superb string-handling facilities and has the ability to translate back and forth between decimal and hex. There is never any garbage collection with Wang strings, and binary object code can be loaded and manipulated as string arrays since both formatted and unformatted string arrays are allowed.

Mr. Tripp's point 4 (the large program problem with line numbers used as labels for GOSUBs and GOTOs) has, I understand, been licked in some extant BASIC interpreters. In any event, it is clear that that problem can be licked, as well as his points 1 and 2. Now that Mr. Tripp has brought the matter to my attention, it is clear that a good BASIC should support cursor commands such as CURSOR (R,C) to place the cursor, READ CURSOR TO (R,C), and even READ CURSOR LAST TO (R,C). The meaning of the first two statements is (I hope) evident, and the last means 'where was the cursor when the last CR/LF was input?'. All of these can be readily implemented.

This brings us to Mr. Tripp's sixth and final point — the slowness of the interpretation process. All of us know (presumably) about the use of BASIC compilers to fix this problem and also about the inconvenience of the compilation process. However, there are two ways to cure the problem, which do not involve a separate and distinct compilation. The obvious one is to use a newer, faster processor chip than the 6502 or 6809 with which MICRO readers are familiar.

The second method is to write a BASIC in which the stored program is essentially a compiled BASIC but where tables of variable names, tables of labels and such, are used along with an interpreter during the editing phase only, which makes the operator believe that the BASIC is fully interactive.

Even now, in the industry-standard Microsoft BASIC, a keyword such as GOSUB is not stored as ASCII "G, O, S, U, B". Instead, those ASCII characters are replaced with a single byte. Then, when listing the program, the computer makes the operator believe that the program does indeed store the GOSUB command as five ASCII characters. My suggestion is that we (considerably) extend this use of the computer to obscure from the operator the exact form of the stored program while persuading the operator during program input and editing that the stored form is conventional.

All of the complaints raised by Mr. Tripp are valid in some BASICs. Some of those complaints have already been fixed and all of them can be fixed. What we need is a properly written BASIC.

Hal W. Hardenbergh
Digital Acoustics, Inc.
1415 E. McFadden, Suite F
Santa Ana, CA 92705

Updates and Microbes

MICRO Calc Mistake

Listing 1 in MICRO Calc (58:47) by Loren Wright contained several errors. [CH] indicates the HOME key. The following lines contained errors or omissions. Note that line 115 is a new line.

```
20  CR$=CHR$(13):DL$=
    CHR$(20):RB$="[RVS]
    [OFF]":BL$="[20
    SPACES]":DI$="◆
    [OFF][CL]"
110 S$=S$(LL):IF RIGHT
    $(S$,1)="?"THEN
    PRINT"[RVS]"BL$CR$
    "[CU]";
115 PRINTS$DI$;
180 IFT$="←"THENS$(LL)
    =S$:GOSUB 5000:LL=
    1:GOTO110
2010 IFT$="@"ORT$=CR$
    ORT$="[CD]"ORT$=
    "[CU]"ORT$="←"ORT$
    =DL$ORT$="[CLR]"
    THENRETURN
8510 PRINT"[CLR]";:FOR
    I1=1TONL:S$=S$(I1)
8520 PRINT"[YEL][CD]"
    S$"[RVS]"LEFT$(BL$
    ,20-LEN(S$)):NEXT:
    PRINT"[CH][CD]";:
    RETURN READY.
```

Corrections for Apple Micro Calc
61 THEN
65 ''SS'' instead of ''BB''
1000 (Include) POKE 216,0

Model Rocket Line Change

In listing 1 of ''Model Rocket Simulalation in BASIC'' by David Eagle (56:31), substitute ↑ for the ± in line 120.

Missing Line

I caught an omission in the program BANK from ''Discrete Event Simulation in Pascal'' in the January issue (56:21). Add the following line: newtime := round(rnexp(1,uariv)) + time; just before the line ''schedule ('d',newtime);'' in 'service'.

Jeff LaBarre
Santa Monica, CA

Review Correction

The review of WP 6502 Version 1.3a from Dwo Quong Fok Lok Sow (57:99) needs a slight revision. Replace the last sentence in the documentation section with: ''The disk is supplied with a number of text files already in place, complete with errors to be corrected with the editing features of WP 6502.''

Earl Morris
Midland, MI

MICRO™

MICRO™
Editorial

The New Wave

When we decided to run a feature on the "new wave" of computers almost a year ago, we knew several new systems with different types of configurations would be entering the market in 1983. We believe these computers are an important trend and — because of the low prices — will help draw more people into the world of microcomuters. As discussed in last month's editorial our latest section, The Learning Center, will offer new users programming techniques and applications for these systems.

But we want readers who have been with us over the past few years to know that we are going to continue to support them. Apple and PET owners will still find high-level information and programs. And we'll also offer *advanced* material for VIC-20, TRS-80 Color Computer, Commodore 64 and Atari users, in addition to The Learning Center.

There is no doubt that the nature of the microcomputer world is changing — it's no longer only for the highly technical computerists using complicated and expensive systems. Technology has allowed manufacturers to produce sophisticated yet easy-to-use computers at an inexpensive price. No longer is there the huge gap between strictly game machines and intimidating computers. There are many varieties in the middle. Micro will keep you up to date on future "new wave" systems and how to use them.

MICRO Moves North!

We're still tripping over boxes and packing foam, but everyone agrees that our move to southern New Hampshire was a good idea. Our brand new offices among the pine trees will provide an inspiring setting to continue to mold MICRO into an exciting magazine.

Of course, as with any move, we (and our readers, advertisers, and authors) suffered thorugh quite a bit of confusion. For instance we were in the new building for three days before the phones actually worked. So if you tried to call our old number you were connected with the few souls left in Chelmsford who had no information except our new number — which wasn't working yet! We apologize for any inconveniences.

A big reason for the trek north is that over half the staff lives in southern New Hampshire. Commuting time has been cut considerably, which gives us more time to devote to the magazine. So, as soon as the phone lines are connected and the last box gets unpacked, we'll all be smiling and back to work as usual.

Our new address is: 10 Northern Blvd., Northwood Executive Park
P.O. Box 6502
Amherst, NH 03031
Our new phone number is: (603) 889-4330

Marjorie T. Morse

Marjorie Morse

MICRO™

PET Vet

Loren Wright

One convenience offered by Commodore computers is the load-and-run function of the shifted RUN/STOP key. With BASIC 4 PETs and CBMs, pressing the key loads the first program on drive 0 of the disk and immediately RUNs it. With the VIC, Commodore 64, and pre-BASIC 4 PETs, pressing the key causes the next program on tape to LOAD and immediately RUN. Doing this from within a program is a different story, though.

There are many applications for LOAD-and-RUN programming, and even more ways to accomplish it. You can load one program from another without destroying the variables defined in the first. This is a technique described by Jim Strasma in Part 1 of his "It's All Relative" series (MICRO 55:37, December '82) under the topic of chaining. Strasma's application was a mailing list program that used one start-up module to define a number of constants, with some based on the current equipment configuration. This way he could avoid redefining these in every new module. The technique also allowed him to pass parameters from one module to another. For more details see Strasma's article.

It is also possible to perform this kind of chaining without changing the values of pointers if you make sure the first program is always longer than the second. If this isn't the case, then add some extra REM statements at the end of the first program. This technique is essential if you have a program that is too long to fit into memory at once. Just find a convenient place to split it, make the first part longer than the first, and include an appropriate LOAD statement for the second part. Variables do not have to be redefined and are passed intact to the second program.

We now have techniques for chaining programs without destroying variables. However, there are situations when neither of these techniques will do the job. One is when you want to load a machine-language program at the top of memory, followed by a BASIC driver. Another is when you need to do a NEW before running the second program. In both cases we need the pointers set to their default values, but the programmed LOAD leaves them as they were for the original program. Also, since a NEW causes the program in memory to disappear, you can't very well LOAD a new program if the LOAD instruction no longer exists! *(Continued on page 14)*

PET Vet Listing

```
10 Q$=CHR$(34): NC=158: KB=623:
   DN$=",8": PO$="■■■■"
20 PRINT"■": FORI=1TO3: READNA$(I
   ): PRINT NA$(I)"■": NEXT
30 INPUT"■INPUT PROGRAM #";N
40 PRINT"■NEW"
50 PRINT"■■LOAD"Q$NA$(N)Q$DN$
60 PRINTPO$"RUN"
70 POKE NC,4: POKE KB,19: FORI=1
   TO3: POKE KB+I,13: NEXT
1000 DATA PROGRAM 1,PROGRAM 2,
     PROGRAM 3
```

PET VET *(continued)*

Having just finished preparing MICRO's new book "Mastering Your VIC-20" for publication, I wanted to be able to demonstrate the programs at the West Coast Computer Faire. The book comes with a cassette, but obviously that is not the appropriate medium for a demonstration. I needed a menu program that would display a list of the programs on the screen and automatically LOAD and RUN any program selected. I tried making my menu program longer than the longest program by adding REMs at the end, but that didn't leave enough room to store the variables created by other, shorter programs. Also, several programs involved lowering the top-of-memory pointer to protect RAM-defined character sets. When a longer program is loaded after one of these, there isn't enough memory available. Obviously (since nothing is impossible), a different technique was required. I read about this technique in the early PET literature, but rather than digging through back issues, I decided to rediscover it myself.

Programming the Keyboard Buffer

This technique has a number of applications besides automatic loading, including having a program add lines to itself or write a whole new program! One commonly used variation is to convert machine language into BASIC DATA statements.

The addresses involved are different for the different Commodore machines, so consult the table for the Addresses for your machine. Below is a demonstration program for loading one of three programs. (It is up to you to make sure there actually are programs by those names on the disk!)

The Table

Function	Variable	BASIC 1	BASIC 2/4	VIC/C64
No. of chars. in keyboard buffer	NC	525	158	198
Start of key-board buffer	KB	527	623	631

Program Description

What the program accomplishes is very simple. First a NEW statement is executed, then the LOAD, and finally a RUN — all as if they were typed in the direct mode. The keybaord buffer holds up to ten keys in sequence, and the PET processes them one at a time in the order they entered the queue. The program puts the commands on the screen in the proper positions. Line 70 puts a 4 in address NC, telling the PET that there are four unprocessed keys left in the buffer. Then four characters, a home and three returns, are put into the first through fourth positions in the buffer. The program is done, but the PET must process those four characters. The effect is as if you typed the commands yourself and then pressed the RETURN key.

Change the values of NC and KB according to your Commodore computer. If you are working with cassette, you are limited to the next program on the tape. To experiment with the automatic load you will also have to change DN$ to ",1" or " " and add three more cursor-down's to PO$.

MICRO™

MICRO™

From Here To Atari

Paul S. Swanson

This month's column is based largely on a letter I received from Jose H. Nieto, a MICRO reader in Venezuela. His letter outlined a few questions that are commonly asked in reference to specific applications. His application included drawing geometric shapes.

Mr. Nieto's first question was about monitors. When drawing shapes in mode 8, you will get color artifacts on any compatible monitor or television because of the mechanics of the signal, although some monitors will reduce the artifacts a bit. However, using the computer for plotting single points to define geometric shapes does not really require color in mode 8. This mode actually is a one-color mode, allowing two luminances. I would prefer to do such work with a good black and white or green phosphor monitor. The image is much clearer than that obtained by either a television or a color monitor.

The Atari 800 computer can be connected to almost any raster scan monitor that uses the same sync signals a television uses. The Atari 400 cannot be connected to a monitor; its only video output is combined with its only audio output and modulated for television use.

The second question in the letter deals with connecting an Atari 800 computer to a stereo amplifier. First, the sound output from the Atari is monaural. You can still use a stereo amplifier, but you will not get stereo sound. The jack on the side of the Atari 800 has connections for the monitor and the sound channel. This jack has five pins on it. The ground pin is the lowest pin on the jack (pin 2). This should be connected to the ground wire of anything you connect to this jack. The "live" connection for sound is pin 3, which is the top pin on the left side as you look at the jack. If you connect both audio input lines to this pin (the "live" from each channel) your stereo amplifier should work just fine. The proper place in your amplifier to connect the Atari output is the auxiliary input or the phonograph input (if there is no auxiliary input).

For color monitors, pin 4 is the proper place to connect the live video wire. Black and white does not require the color information and I have had better results connecting those monitors to pin 1, which is composite luminance. Pin 4 is composite video, which will carry the color burst required for color pictures, but not required for black and white.

Another question in Mr. Nieto's letter refers to the use of a PRINT #6 command with a mode 0 screen. This will normally leave inverse video spaces at the end of each line. If you want to access a mode 0 screen with PRINT #6, don't use a GRAPHICS 0 statement. Instead, declare the mode with the following statement:

```
OPEN #6,0,0,"E:"
```

The screen is opened with a GRAPHICS statement as the screen handler (device S:). Using the screen editor, device E:, will eliminate the white squares. The white squares you get when you use GRAPHICS statements with PRINT#6 is actually the cursor, unreversed after it is moved. Use the device E: and you won't have to worry about it.

Since the white squares left behind are unreversed cursors, there is another simple solution to this. Turn off the cursor right after you declare GRAPHICS 0. This can be done by a POKE 752,1.

When you use GRAPHICS 0 and the POKE 752,1, you can use COLOR, PLOT, and DRAWTO with a mode 0 screen. Try, for example, the following short program:

```
10 GRAPHICS 0
20 POKE 752,1
30 PRINT
40 COLOR ASC("A")
50 PLOT 0,0
60 DRAWTO 10,10
```

Lines 10 and 20 open the screen and shut off the cursor. Line 30 eliminates the single white square that would be left if the PLOT were to be executed without it. The COLOR in mode zero is the character to print. The PLOT and DRAWTO will set a diagonal line comprised of A's on the screen. Eliminate line 30 to see the cursor that it eliminates. Change the COLOR statement for other characters. The control characters will print as characters in this method, so you can declare COLOR 28 in line 40 and get a line of up arrows. Even the ESC character works (COLOR 27). Only RETURN (COLOR 155) gives unusual results.

One last question concerns the graphics available on printers. There is no redefinable character set on the Atari 820 printer and I know of no graphics available on it either. The printer I use is an Epson MX-80 with Graftrax Plus. The character set on that printer is also not programmable, but you can address all of the dots on each line individually in its graphics modes. It is rather simple to write a screen dump program for the Epson, if you dump the screen sideways. If you have an Epson with any Graftrax option, read the graphics section of your Epson manual and you will see how this can be done.

Both of the 800 series Atari 40-column printers also have graphics capabilities. If you do not need the 80-column width, the suggested $299.00 retail price of these printers may be worth considering.

New Hardware

In other news, Atari has a few products to add to their computer line. In addition to the well-publicized 1200, Atari has a new Program Recorder, model number 1010, that is scheduled for first quarter (it should be available by the time you read this). The 1010 is like the 410 except it has a case design more similar to the new look of the Atari 1200. Also reported is that this new recorder has an improved mechanism. It will be in the same price range as the Atari 410.

In the second quarter, two new printers are scheduled. The Atari 1020 is listed as a 40-column color printer. It actually functions like a drum plotter with colored pens. It uses the 40-column sized paper and can be addressed as a printer. The characters printed to it are plotted in normal, compressed, or expanded mode, as controlled by the program. The suggested retail price for the 1020 is $299.00.

The Atari 1025 is a new 80-column printer that does not require the 850 interface. It reportedly does not have graphics capabilities. Suggested retail price for the 1025 is $549.00.

Write to Paul at 97 Jackson St., Cambridge, MA 02140.

MICRO™

MICRO™ CoCo Bits

by John Steiner

The winds of change blow surely through the computer industry, and the Color Computer is not to be left behind. A newer, more powerful DOS is rumored to be on the way, though as I write this there is still no official word from Tandy. The release cannot be too far off as there is mention of an Extended Disk BASIC 1.1 ROM in "The Sands Of Egypt," a newly released adventure game from Radio Shack. There is at least one new command added to the ROM. To execute the program from the 1.1 ROM, just enter 'DOS'. The 1.0 version must use the RUN file command. Tandy will let us know when they are ready.

I noted in a previous column that the color output of my TDP 100 was different from the standard Radio Shack CoCo. This is indeed the case; I have seen the differences in many programs, including the one just mentioned. Computerware says this is a problem on early model TDPs, and they are sending me modification data that will correct the color problem. The new board is now being installed in Radio Shack CoCos, and I am wondering if anyone who has one has noticed a difference from the earlier boards. If you have any information on this, please pass it along.

I am using my TDP and disk system, but with a borrowed disk controller card. The 6809 and controller card failed. The computer has been repaired, and the controller is awaiting a new ROM chip. It should be in any day now. I had been running a tape-based program, and had shut the computer off to reinstall the controller. When I powered up again, the system was down. There is no buffering between the processor and ROM slot, nor is there any between the controller input and ROM. A failure of either unit can easily damage both. As a precaution, be sure the power is off several seconds before reinstalling a card in the slot. Hardware hackers take note: a bi-directional buffer would be relatively easy to build and would allow protection for equipment on the line.

Modems and Things

I received a letter from Paul Van De Plas of Vancouver, BC. He has a Hayes Stack Smart modem, and a CoCo. He would like to use the auto-dial feature and the built-in Morse Code send/receive feature of the modem, but has been unable to access them using the CoCo. If you have any information on using the Hayes modem with the CoCo, pass your information along.

Though I have been using CoCo as a radioteletype terminal in my ham radio hobby, I have never had a telephone modem. When I needed one, I used an old, acoustic coupler, an originate-only unit that I borrowed from a friend. I never felt that the cost of a modem was worth what use I would get out of it. That changed when I happened across a Mura MM100 direct connect, originate/answer unit. The modem is under $100 and takes up only a little space on my crowded desk. I placed it on top of my

(Continued on page 18)

CoCo Bits *(continued)*

radioteletype modem, next to my ham radio, sandwiched between the disk drives. I have had a lot of fun with it, and gained some practical use by accessing the local bulletin board and the educational timesharing network available in North Dakota. I suppose a subscription to CompuServe will be next.

A new accessory available for CoCo is a video monitor output adapter. This device connects to CoCo's RF output and delivers a high-quality signal to a standard video monitor. There is no modification required to the computer, and it should provide an excellent video signal. Flex, in its 64 character-per-line mode, will be the test of this promising new accessory. Though color graphics are nice, the color TV is a poor monitor at best. Any improvement along this line would certainly be welcome.

I have a new drive unit on line and am happy with it so far. It is an MPI from Frank Hogg Labs. Though noisier than my Radio Shack drive and slightly longer, its color and style fit well with the system. The drive, a 40-track unit, allows me to read Flex files written on 40-track disks.

Flex allows you to specify different system and working drives. For example, the DOS will keep track of the fact that drive 0 might be 35-track double density, while drive 1 is double sided, double density, 80 track. That information is stored in the directory, which is always written in single-density format. By allowing different drive styles, a much greater compatibility is possible.

While on the subject of Flex, I must comment on the ease of getting printed information from files. Flex allows all commands to be prefaced with a P, which will send all file output to a printer. Flex allows you to change the baud rate from 110 to 9600. You may choose a carriage return plus line feed, or the standard Radio Shack carriage return where the printer must send its own line feed. You may set the number of stop bits as well.

Service Manuals

In the last few weeks, I have received several letters regarding the availability of service manuals for the CoCo disk drive units. Radio Shack Computer Center managers are informing people that manuals are not available. There are two manuals covering the disk units — one covers the drive and the other the drive controller card. The part numbers for the manuals are MS-260-3022 and MT-260-3022 respectively. The pair costs less than $10.00. According to a local store manager, these manuals are available from Radio Shack National Parts Division. If you provide your dealer with the above data, you should have no trouble obtaining them.

Next month I will have more comments on using Flex, Star-DOS, and other CoCo operating systems.

Contact Mr. Steiner at 508 Fourth Ave. NW,
Riverside, ND 58078

valFORTH™ for Atari* 400/800

Professional Software for the Hobbyist

VALPAR INTERNATIONAL

FORTH has been used for years by ATARI* and others in programming their arcade games. FORTH is *fast*, 15-20 times faster than BASIC, and can make use of every capability for your computer. And it's no longer the province of the professional programmer! With valFORTH and the additional packages described below, you can create programs in an afternoon that would previously have taken weeks of hard work!

WHAT? YOU DON'T ALREADY KNOW FORTH?

Then take advantage of our special offer on *Starting Forth* by Leo Brodie. Widely acclaimed as the best book available on the subject, this entertaining treatment of Forth will make you comfortable with this exciting language quickly and easily. When you order *Starting Forth*, you get a free copy of our "Notes for the valFORTH User," including references, by page, to *Starting Forth*.

These are the utilities developed by Valpar International's software specialists and used to create our commercial software products.

valDOS

NEW! By Popular Demand!

valDOS. Now you can use your valFORTH system to read and write normal Atari DOS Files created with valDOS or other sources. Also includes valDOS File Editor for creating and reading FORTH Source Code without "screens." (Not a general file editor.)

(Over 25 pages of documentation. Requires valFORTH.)

valFORTH T.M.

Package contains: fig-FORTH kernel with mathematical and stack operations machine-coded for higher speed than normal fig-FORTH; line editor AND screen editor, debugger, sound and graphics commands, floating point, advanced 6502 assembler, diskcopiers, and much more! (Over 110 pages of documentation)

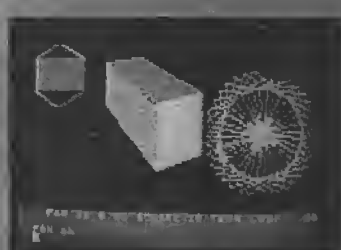
GENERAL UTILITIES AND VIDEO EDITOR

Utilities: 4 array types, 4 case types, text on graphics 8, extensive string manipulation and keyboard input, STICK & PADDLE, randoms, bit manipulation, and much, much more.

Editor: *fast, powerful, complete* valFORTH screen editor 1.1. A professional-quality tool that makes editing a pleasure.

(Over 60 pages of documentation. Requires valFORTH.)

PLAYER-MISSILE GRAPHICS, CHARACTER EDITOR, & SOUND EDITOR



Player-Missile: Create, move, color, change images of and bound players and missiles with high level commands. Full support of 5th player, multicolor players, etc. etc. All critical sections in machine code.

Character Editor: Compose character sets with joystick. Simultaneous display of created characters. Make images for players and missiles.

Sound Editor: Simple independent control of all four voices (one joystick per voice) and audio-control register. Create any single-setting sound with graphical and tabular readout.

(Over 35 pages of documentation. Requires valFORTH.)

Coming Attractions:
* TARGET COMPILER
* 3D WORLD

For more information *fast* and a 4TH ♥ IF HONK THEN bumpersticker, send 25¢ and a self-addressed, stamped envelope.

DISPLAY FORMATTER

Fast, simple creation of all types of display lists, with automatic 4K boundary jumping. Automatic or user controlled memory allocation, and formatting for horizontal and vertical scrolling, and display list interrupts.

(Over 30 pages of documentation. Requires valFORTH.)

TURTLE & valGRAPHICS AND ADVANCED FLOATING POINT ROUTINES

All graphics modes supported — even GTIA and "7+". Draw and fill commands *faster* and *smarter* than Basic. "Turn-toward" for "chasing" and vanishing point effects; point labeling, etc. Also SIN, COS, ATN, ATN2, etc. added to floating point.

(Over 35 pages of documentation. Requires valFORTH.)

Text Compression and Auto Text Formatting

A unique, two-part utility!

- Text Compression allows the packing of text into much less space than normally required. Useful for wordy Adventure games, "artificial intelligence," etc.!
- Auto Text Formatting takes both normal and compressed text and routes it to the video screen "windows."

(Over 20 pages of documentation. Requires valFORTH.)

valFORTH alone requires 24K
valFORTH plus one or more packages requires 32K minimum
Memory requirements include 10-12K working space
All products are now on non-protected disks
Over 350 pages of detailed documentation!

valFORTH 1-1	$59.95
General Utilities and Video Editor	49.95
Player Missile Graphics, Character Editor, and Sound Editor	49.95
Display Formatter	39.95
Turtle and valGraphics and Floating Point Routines	54.95
Text Compression and Auto Text Formatting	39.95
valDOS and valDOS File Editor	59.95
Heavy-Duty Professional Binder (provided free when ordering 3 or more packages at one time)	13.00
Starting Forth	15.95

Plus Shipping and Handling
VISA and MASTERCARD accepted

VALPAR INTERNATIONAL
3801 E. 34TH STREET
TUCSON, ARIZONA 85713
Call Toll-Free 800-528-7070   In Arizona call (602) 790-7141

Principal Software Authors: Stephen Maguire and Evan Rosen

THE ATARI 1200XL

by Paul S. Swanson

Differences between the Atari 1200XL and earlier 400 or 800 computers are significant. The most obvious difference is the physical appearance. The case design is sleek, particularly in comparison to the 400 and 800 systems. The keyboard is similar, but the two most troublesome keys have been moved off the main keyboard. These two keys are BREAK, which can accidentally stop a program running on a 400 or 800 if you miss BACK S by one key, and the Atari key, which inverts the video output if you miss the question mark by one key. These two keys have moved up to the metal strip above the keyboard.

There have been five keys added to the function keys available on the 400 and 800. One is the HELP key, which can be strobed by a program to sense user distress or any other defined symptom. There is also a built-in demo program in the 1200XL that uses the HELP key to control its functions.

The other four keys, F1 through F4, can be used in combination with SHIFT and CONTROL (renamed from the 400 and 800 CNTL) to provide a total of 12 keyboard-initiated functions. With no SHIFT or CONTROL, these keys act like the arrow keys do with CONTROL down, so they contribute single keystroke cursor control. With SHIFT down, the four keys add four additional cursor controls. SHIFT F1 will move the cursor to the upper left corner of the screen, SHIFT F2 to the lower right, SHIFT F3 to the beginning of the physical line, and SHIFT F4 to the end of the physical line.

CONTROL with these four function keys creates more involved functions. CONTROL with F1 locks out the keyboard, which can be restored by pressing the same combination again. CONTROL with F2 turns off the video display, which is restored by pressing any keyboard key. CONTROL with F3 toggles between keyboard click and no keyboard click. CONTROL with F4 changes the character set to the new built-in international character set.

Atari omitted the cartridge door from the 1200XL. This welcome change circumvents the physical weakness of the 400 and 800 case designs. The only part on my 400 or 800 that has ever failed is the cartridge door hinge, a problem I don't anticipate with my 1200XL. The cartridge has been moved to an open slot on the left side of the computer. This should provide many possibilities. With no cover containing the cartridge, larger cartridges (actually external to the computer) can be constructed that bypass the restrictive size constraints of 400- and 800-compatible cartridges. A slight alteration of dimensions on the cartridge slot may make some third-party cartridges incompatible with the 1200XL, but all of the Atari-produced cartridges will work easily.

The rest of the external accessories on the 1200XL are identical to the ones on the Atari 800 except that there are only two controller jacks on the 1200XL and four on the 800. There is a monitor jack as well as a modulator for connection to a television, a serial port jack, and the selector for channels 2 or 3.

Operating System Improvements

As indicated by the four function keys, there have been operating system changes. The international character set, implemented by CONTROL with F4, replaces the keyboard graphics characters with a set of 29 international characters with diacritical marks with the letters as well as the British pound and inverted exclamation point. As with the 400 and 800, you can also invent your own character set. L2, which is a LED indicator on the keyboard, lights when the international character set is selected.

The keyboard click is no longer produced by the keyboard. All of the functions are still possible, but they are channeled through the television speaker. When the keyboard click is suppressed, all of the other sound channels still operate normally.

Turning off the video display can speed up processor time by about 25%, which is done by eliminating DMA (Direct Memory Access). The real-time clock (which does exist on all Atari computers, contrary to Commodore's advertising claims) continues to run, so the vertical blank interrupt is still in operation with this function implemented. From timing loops that I ran on both the 1200XL and on the 800, it appears that the vertical blank interrupt has also been rewritten and runs faster on the 1200XL.

The additional memory that was added to bring the 1200XL to 64K is bank selected into the 6502's address space. This implementation excludes direct access to the additional memory from BASIC; however, it can be utilized by machine-language programs and subroutines, including those called from BASIC.

Although I have no documentation on it, I believe the video output circuitry has been improved on the 1200XL. The colors are brighter and the contrast is sharper. In switching from normal television viewing to using the 1200XL in the normal text mode, I have found it necessary to turn down the contrast significantly or set color registers 1 and 2 closer together in luminance.

The locations used to read and write controller ports 3 and 4 on the 400 and 800 are redefined on the 1200XL. These locations (Port B) are used to implement, under program control, the new features available from the four new function keys. The other new LED, L1, is lighted when the keyboard is disabled. Writing to the Port B registers controls L1 and L2 the way the function keys would. Port B also controls the bank selection options.

Another operating system enhancement is the addition of four graphics modes. All four of these new modes are actually available on the 400 and 800, but can be declared by a GRAPHICS statement from BASIC on the 1200XL because they are operating-system supported. To implement them on the 400 or 800, you must build your own custom display list.

The four modes include two character graphics modes and two map modes. You must build and locate in

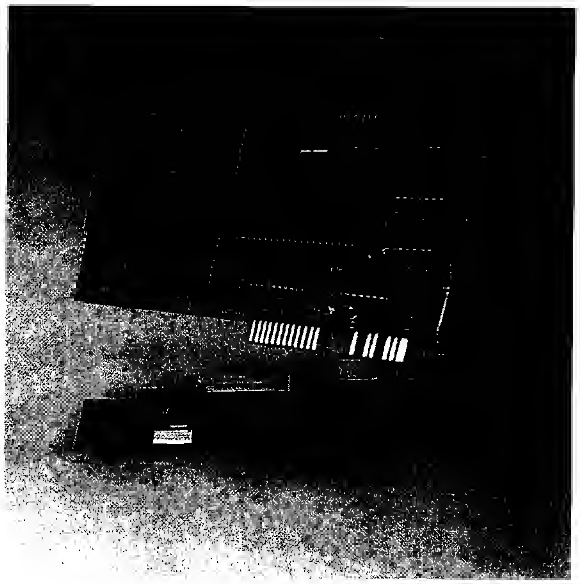
MEGAFLEX® ABILITY

You Pick The Disk System, MegaFlex Controls It!

WITH SOFTDRIVERS FOR A FLEXIBLE FUTURE!

MEGAFLEX—a universal floppy disk controller and modern alternative to the Apple drive system offering increased storage, improved reliability and . . . FLEXIBILITY.

Enjoy megabytes of online storage with your choice of micro, mini, or maxi drives—or even 6Mb with the Amlyn cartridge pack! Ideal for high-capacity storage now, winchester-disk backup later.



The MEGAFLEX secret is to autoboot softdrivers that match the needs of your drive system. All hardware functions are software-controlled. MEGAFLEX can match new drive capabilities without hardware changes. Drive-dependent ROMs have been eliminated.

APPLE *///*? OF COURSE!!

MEGAFLEX is compatible with BASIC, CP/M, Pascal, VISICALC, SOS and DOS-emulation on the Apple III, Apple II, Franklin Ace and Basis. All language features and operating system commands (LOAD, BRUN, etc.) are standard. If you can operate Apple drives you can operate MEGAFLEX! Your Apple software will run without modification too.

BRIDGE THE APPLE FORMAT BARRIER!

The MEGAFLEX diskette does what Apple's cannot—read and write diskettes from other computers! Software-controlled industry-standard IBM 3740 or System 34 type formats allow the MEGAFLEX library of reformatting software to read and write Altos, Radio Shack, Osborne, and IBM PC diskettes. (Call for the latest software details.)

MORE STORAGE, MORE UNIVERSAL FEATURES, LOWEST COST

MEGAFLEX with 8" maxi or high density 5.25" minis gives you 1.2 Megabyte of formatted data per diskette for 8 times the file and data size!

MEGAFLEX offers flexible software choices:

- data rate (250/500 Kbits per second),
- single and double density recording, and
- single/double sided drive operation (max 4 drives).

MEGAFLEX has the lowest chip count of any controller today! This means less power, a cooler Apple and better reliability.

Lowest price, highest performance, that's MEGAFLEX!

MEGABYTER™
A Division of SVA

11722 SORRENTO VALLEY ROAD
SAN DIEGO, CA
(619) 452-0101
TWX 910-335-2047 APPLE TWO SDG

TRADEMARKS CP/M—Digital Research

memory a character set when using the character graphics modes. The character set information is interpreted the same way bytes on a four-color map mode display are interpreted except that, when the character is printed in inverse video, bit combination 11 uses register 3 instead of register 2. Although much of the literature from Atari that deals with these two modes calls them four-color modes, they are actually five-color modes, restricting each character to four of the colors. The colors for each character are background (register 4), register 0, register 1, and either register 2 or register 3.

The new map modes are 160 × 192 dot displays. One is a two-color mode and the other is a four-color mode. The addition of these modes means that all of the internal IR modes except one are now operating-system supported. The one that isn't is a variation of mode 0 using ten scan lines per character line instead of the eight used in mode 0. This enhances the appearance of the characters by allowing two extra scan lines for the lower-case descenders. Judging by the number of programs that I have seen using this mode, the demand may dictate that Atari may never bother to support it in their operating systems. It can produce nice output, but mode 0 is usually considered good enough.

Concerning the Competition

This new offering by Atari is not without competition; for example, the Commodore 64. The Commodore system is a sophisticated assembly of electronics, implementing several of the same features implemented by the Atari computers. The Commodore has more and larger sprites than the Atari's player/missiles. The Commodore has three sound channels as opposed to the four in the Atari computers, but the Commodore has more controls for attack and decay as well as finer selection of frequency. However, the sound channels on the Atari computers are more capable of producing sound effects.

There are 16 colors available on both the Atari line of computers and the Commodore 64, but the Atari computers also give a selection of luminances. Correctly written graphics routines on the Atari have much more depth than routines written on computers such as the Commodore 64 and Apple because of these luminances.

The Atari display output is also more in tune with the television — the normal video output in the home where these computers are intended to be. There are only 160 color clocks across the visible area of a standard color television. This makes it physically impossible to have a selection of two different colors for the text mode if 40-character lines are used. If you have a Commodore 64 and you try to select certain combinations of color for the text mode, evey other letter will be illegible because the television cannot display the number of color clocks required. The Atari text mode uses one color with two luminances for the characters in the text mode with at least two dots for the width of each dot or line in the characters. This makes the Atari characters appear shorter and fatter than the characters used on other computers, but also makes them much more legible because the method used on Ataris is more consistent with the way the television processes the signal.

New Peripherals

No drastic physical change in appearance would be complete without color-coordinated peripherals. Already available is the Atari 1010, which is an upgrade of the Atari 410 Program Recorder (cassette drive). The case is consistent with the design of the 1200XL and the internal circuitry has undergone a few improvements. Some existing cassette-based programs may give you problems on the 1010 and 1200XL combination, but Atari includes a notice in the literature supplied with the 1200XL that promises to solve any problem you may have with APX cassette programs, free of charge, of course.

Available in the second quarter, possibly by the time you read this, are two new printers. The Atari 1020 is advertised as a 40-column color printer. It uses multicolored pens to plot letters and/or graphics. The Atari 1025 is an 80-column printer that does not require the 850 interface. I have been told that it does not have graphics nor some of the other fancier options afforded by third parties (I'll be keeping my Epson MX-80 with Graphtrax Plus), but not requiring the 850 makes its suggested retail price of $549 very attractive for 80 columns of printed output. Other 80-column printers may have similar prices, but require an additional $200 for the 850. Other new peripherals are expected in the near future.

Looking Ahead

Atari will be announcing new computers in June. Although not much has been released officially, two new computers, called the 600XL and 800XL, are expected. The new computers will have BASIC as a built-in feature according to the sources I have tapped for information.

Rumors concerning the new computers should not be trusted, but are interesting. I have heard mumblings of built-in speech synthesizers and modems. If these prove to be true, I would say that Atari is making real progress in advancing the quality of home computers. The internal graphics circuitry used in the Atari seems to be, by far, the most advanced available in any computer in the same price range.

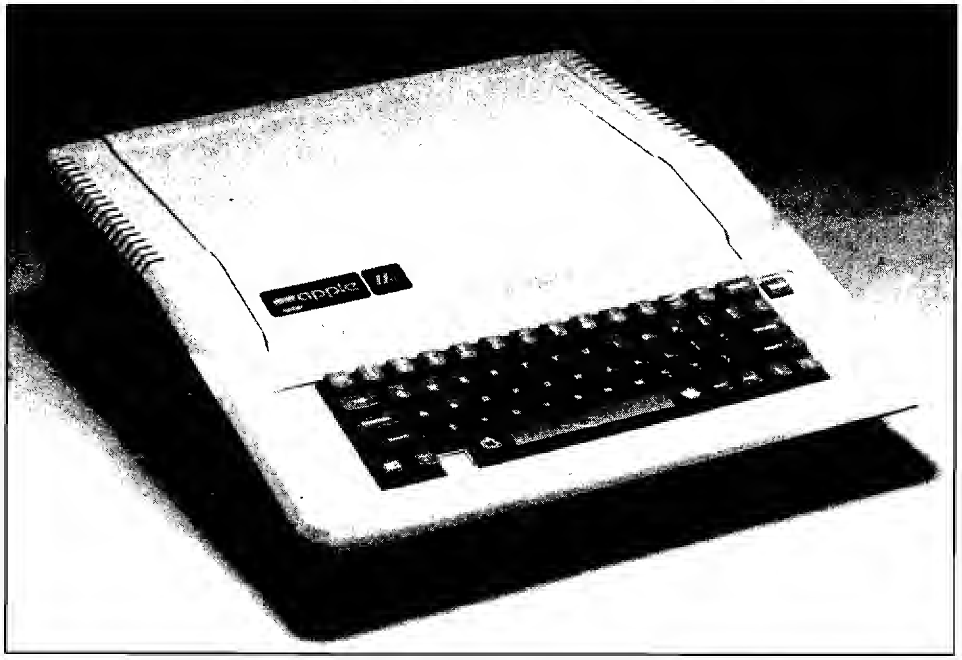
Third-party suppliers should become very active in producing new peripheral devices for the 1200XL. Although this computer has been criticized for being a "black box" with no open circuitry that the true hacker can start cutting up and rearranging (probably one of the stronger selling points for the less sophisticated Apple computer line), the 1200XL actually offers something not easily available on the older Atari computers. The open cartridge slot has 13 of the 16 address lines with a select line governing the other three, all eight data bus lines, and several other control lines available. Little imagination is required to think of what can be done with this slot.

You may contact Paul Swanson at 97 Jackson Street, Cambridge, MA 02140.

MICRO

New Microprocessors for Your Apple II

by Phil Daley and John Hedderman



Do you feel left out? Does everyone seem to have a new computer but you? You can have a new computer without selling your old one.

There are several add-on boards for the Apple II that increase its capabilities, provide new features, teach you new languages, and allow you to run software written for processors other than the 6502. We discuss several of these boards: the Excel-9 6809 by Norell Data Systems, the Mill 6809 by Stellation II and the Dtack Grounded 68000. We have also been working with an Apple III and will present a short summary of its features.

Let us start off by saying that we like the Apple II and the 6502 microprocessor and have not found serious deficiencies in either. However, after using a 6809 microprocessor, you cannot help but be impressed with the increased capabilities of the 'cadillac' of 8-bit microprocessors. Just having 16-bit registers makes a large difference in programming ease, and being able to long branch to any portion of memory makes writing location-independent code a snap. When you add on the power of a structured operating system like OS-9, the results are dramatic. Certainly no one will argue the advantages of a 68000 microprocessor. With 32-bit registers and a 12.5 MHz clock, the speed differential on identical code is enormous. When you add the programming ease and addressing capability of the 68000, you have amazing potential.

TSC's Flex

The Excel-9 board comes with the complete FLEX manual from Technical Systems Consultants and is well written, easy to understand and complete. Unfortunately the Apple documentation from the Norell people was written in Japanese and translated into English, or at least that's the way it reads. The documentation is somewhat sketchy but gives a satisfactory explanation of the setup and operation of the Excel-9 board.

The first step in using a new disk is to always make a backup before something disastrous happens. To format a Flex disk, a program called NEWDISK is executed. We typed in 'NEWDISK 2' to refer to formatting a disk in Apple drive 2. Unfortunately that did not work, as a 'DRIVES NOT READY' error message was displayed. Apple normally calls the drives in each disk controller card '1' and '2'. The slot number of the controller card and the drive number of the disk are necessary to completely specify the desired drive. Flex numbers the drives, starting with '0', and continuing in sequence. The Apple drive #1 is considered as Flex drive #0 and Apple drive #2 is Flex drive #1. This is probably less confusing than the CP/M system of Apple drive '1' being CP/M drive 'A'.

Typing 'NEWDISK 1' sucessfully formatted a blank disk. All commands in Flex are executed by typing the program name. The documentation says to make a backup the proper command is 'BACKUP'. When we typed 'BACKUP', a 'NOT FOUND' error was received. A quick CATalog of the disk revealed a COPY command, and by typing 'COPY 0,1' produced a back up of the Flex master disk. Next we 'LINKed FLEX' to produce a bootable disk. The bootstrap program has to know physically where the system DOS resides on the disk, and the program LINK stores the starting track and sector of FLEX.SYS in track 0, sector 1 so that upon booting a disk, the boot program can load Flex into memory.

All of the normal Flex utilities are included on the system disk, and they all work as specified in the Flex manual. Using the LIST, ASN, CAT, etc., commands produced the standard Flex results. To try the 'P' command, you attach a regular printer cable to your interface and printer, we used an Epson MX-80 with the Apple serial card. Typing 'P,CAT 0' produced a hardcopy catalog list.

To delve a little deeper into the innards of Apple Flex, a sector read utility was used to examine the backup copy of the Flex system disk. The track and sector linking that Flex uses to access the sectors in an interleaved fashion seemed to be rather haphazard and not sequentially as on other 6809 computers. It appears that the Apple NEWDISK writes the sectors in a physically sequential process and then links them ahead four or five sectors to optimize access speed, whereas Flex on a regular 6809 computer interleaves the

sectors as it writes them, and then links them in a logically sequential process. This difference prompted us to write a test program (see listing 1), a Flex utility to change the name and number of the Flex formatted disk that is stored in the system information record, using the Flex Advanced Programmers Manual. This program was written and tested on a standard 6809 computer first and then retyped, with no changes, into the TSC Text Editor supplied on the Apple Flex disk.

The program reads the System Information Record (one sector) by calling one of the Flex user routines and, after making the appropriate changes to the information, writes the sector back, referring to its track and sector (Track 0, Sector 3). By employing two different methods of accessing the disk, we gave the system a double test. Each method worked even though the track-sector reference to the disk sector was not the Apple disk sector referenced. Apparently Flex on the Apple translates this difference internally.

The Excel-9 board also has a basic system monitor, which we used to examine some memory. The monitor can do things like Dump, Disassemble, Jump, Compare, etc. One note: when within a monitor command, the documentation doesn't mention the accepted method of return to the command mode. (Reset works, but it's messy.) A {space}{return} seems to work most of the time.

The Excel-9 board works very well and fully implements the Flex operating system on the Apple. It worked without flaw on our Apple II. We couldn't use it on our Revision 7 Apple II without removing the Apple serial card from slot 1. Norell is aware of the problem and is working on a cure. If you want to learn about Flex, the Excel-9 is the board for you.

Stellation II's Mill

Standard software for the Mill, a 6809 board, includes a macro-assembler and debugger for writing machine-language programs on the 6809. Included is a demonstration package for multi-processing — running the 6502 at 1/5th speed and the 6809 at full speed at the same time. Also available is a Pascal speed-up kit to increase the operating speed of Apple Pascal. The multi-tasking operating system, OS-9, is available from Microware and includes a structured basic, BASIC09, a sort of hybrid with Pascal.

The documentation that is packaged with all parts of the Mill system is complete, understandable, and well written. But learning an entirely new operating system requires effort on your part. OS-9 is a 'Unix-like' heirarchical tree-structured system and, as such, takes someone who is used to a simple system like Apple DOS a while to get the hang of it. More on Apple operating systems in next month's magazine.

McMill is a powerful Macro-Assembler that comes with the Mill. It would be very useful for developing software for other 6809-based computers with more limited development systems; eg., the TRS80-C.

To test the assembler for the 6809 Mill, we tried a bubble sort program (see listing 2) from a programming manual. Any normal text editor that makes standard DOS text files can be used to write the assembly file. We used Apple Writer as it is one of the few that is DOS compatible and doesn't word wrap when you don't want it to. After removing the errors due to erroneous labelling, the file assembled correctly. The question arose as to the proper ORiGination, not covered in the manual. The Mill uses addressing which is $8000 relative to the Apple addressing due to conflicting areas of memory. We tried $A000 (assuming that would translate to $2000),which placed it in memory at $A000, the middle of DOS: this crashed the system, so we tried $2000.

This address inversion is the trickiest part of using The Mill. Since having two processors addressing the same RAM memory can cause conflicts, The Mill inverts the most significant memory address bit (A15) prior to being placed on the Apple memory address bus. This means that there are both 'logical' and 'physical' names for each memory location. The physical name is the actual RAM address in the Apple, the logical address is the inverted address that the 6809 thinks its using. In this manner, the conflict with high memory interrupt vectors is overcome. Since the Apple vectors are in ROM, there would be no way to change them for the 6809. Inverting the addresses places the 6809 vectors at the top of the $7000 (physical) addresses.

McMill is a powerful Macro-Assembler that comes with the Mill. It would be very useful for developing software for other 6809-based computers with more limited development systems...

The address inversion also creates another problem. The Assembler creates the code at the ORiGinated address. The output of the Mill Assembler is a text-type file of hexadecimal code. A program called 'LOAD09' is a utiliy on the assembler disk that loads the created file into RAM memory. Unfortunately, it will also try to load the program into ROM or DOS, anywhere the program was ORG'ed. Since this leads to system crashes, it is necessary to ORG the program somewhere in Apple RAM that is available. Then the program must be BLOADed and called at a different address, such as the inverted one. Those of you who have been following closely will see that this places definite constraints on the addressing modes of the original program. The bubble sort program was then rewritten to use only Program Counter Relative addressing to result in a position independent program that can run anywhere in memory. The 6809 assembler makes its very easy to write position independent code, calculating all the relative offsets to the PCR.

A call to the Mill people prompted the suggestion we try $7FFE as the origination point with the first two bytes of zero to overwrite the interrupt vector. That would place the program at $8000 (relative $0000). This is probably the most satisfactory solution.

'MUG' is a program to interactively run and debug 6809 programs. It includes memory dump and disassembly listings, breakpoint insertion, memory search and modify, register change and dump and a calculator mode. It is easily modifiable so that the Mill can reside in any slot. (This is starting to become a problem with all these add-on cards cluttering up our Apple.) The program and documentation are easy to use and, once you overcome the address inverting, easy to understand.

The Mill also has demonstration

programs of the two processors running concurrently. While it is most impressive, it would take some time to learn how to make the best use of this feature.

Another set of programs is the Pascal speed-up kit. This group of files substitutes for the SYSTEM.APPLE and SYSTEM.MISCINFO files on APPLE1: and requires the Mill to be present in the Apple when operating. The directions are clear and established the files on our newly made APPLE1: (now called MILL09: so that we don't get it mixed up). The speed difference is quickly apparent. You can even specify a screen print of which processor is running at the current time, although it didn't show until we turned off the 80-column card.

This is the smoothest running board we tested with the least bugs and best software. It is also the only unit tested that includes software for Pascal. The complete outfit including OS-9, while expensive, is probably the most useful adaptation for the Apple.

Dtack Grounded's 68000

If you are adventuresome and want to experiment with a fast microprocessor, Dtack Grounded makes a 68000 board that is compatible with the Apple. The connection instructions are simple (the board plugs into slot 2) and the demonstration programs are impressive and work flawlessly. A cross-assembler is available from Phase-0, and it appears to work correctly.

Again, it is necessary to use a text editor to make the file (see listing 3) before assembling. The editor must supply carriage returns at the end of each line. ASSEM68K has several nonstandard requirements such as: it uses an 'ASC' pseudo-op instead of 'DC' for ASCII characters, it requires a ';' for comments, and it does not allow an 'END' statement. The Phase-0 documentation does mention the first two, it does not mention the third. The operation of the actual assembly is easy; it only requires a source file name (optional drive number), object file name (default 'source name.OBJ0') and printer slot (default 0 for monitor). The listing format is relatively neat except that comments over 18 characters long wrap around to the next line (on an 80 column printer) and make the listing harder to read.

The assembler disk comes with an upload and monitor programs so that you can learn and try out 68000 code running on the Dtack board. Unfortunately the documentation on the upload and monitor portions is incomplete and hazy. Producing the object code proved to be much easier than uploading it to the 68000 board. We first tried the 'UPLOAD' program supplied on the Phase-0 disk. The program does not handle extra DOS parameters, such as drive number. We had to LOAD, CATALOG, and RUN to read from a second drive.

Next we used the MONITOR program to look at memory. At first we couldn't find the uploaded program and made a quick call to Dtack Grounded. Mr. Hardenburg told us that the upload program loads the program into the Dtack board wherever it is organized. He also gave us a quick memory map, not included in the documentation to help in deciding where to write programs. $0 to $1000 is the bootstrap program, from $1000 to $10FF are the RAM vectors, and from $1100 to $1EFF is the system stack, building down. Therefore $2000 is a safe place for programs with scratch space in the $1100 area.

Since we had ORiGined the program at $1000, it had been overwritten by the RAM vectors. Setting the ORG to $2000, uploading the code and running the monitor program found the program intact and residing at its proper location. The next problem was how to gracefully exit from the program. We tried JuMPing to $122, the monitor input idle. Upon execution, the program returned to the monitor and we dumped the registers. Unfortunately, they showed no results; in fact, except for the Stack Pointer (A7), none had changed at all.

...for those of you who like the 6502 but want a system with more features, there is the Apple III, the most sophisticated 6502 we've seen.

After several program changes/runs and no results, we tried modifying the registers directly with one line programs. When this also gave no changes, we gave up on the Phase-0 monitor program. There seems to be a problem with either the register dump or the execution command.

Using the Monitor that is in the Bootstrap ROM of the Dtack board, while a more complicated procedure, produced better results. We examined the sample programs included with the Dtack board and determined the proper procedure for executing a program on the board. It includes loading the FP BASIC from Dtack on the language card, uploading the file, executing the file and ending with a register dump to view the results (see listing 4). The REM statements are ignored by the Applesoft interpreter and used by the Monitor program as data codes. In this manner, data can be transferred from the Dtack board to the Apple, and vice-versa. There are other CALLs to perform other functions as well. We also discovered the opcode 'TRAP #15', which dumps the 68000's registers and exits the program. TRAPping to vectors 0-14 can provide other exits and routines by changing the vector pointers. In any case, using the Monitor program from BASIC works quite well.

The Dtack board has tremendous potential but requires knowledge of the user. It has the most primitive software at this point, but also the most potential for future development.

Apple's III

For those of you who like the 6502 but want a system with more features, there is the Apple III, the most sophisticated 6502 computer that we've seen. It contains the best elements of the Apple II with improvements of many of its shortcomings, and has additional features not found on many systems in its price field. In as much as we have heard the bad reviws that followed its release, and that we like the Apple II even with its faults, we awaited with trepedation the arrival of the III, and have somewhat procrastinated the testing and experimentation necessary to write this review.

The folks at Apple need not fear, however, as we liked the computer more than we thought we would and, indeed, if not for its price tag, would be tempted to buy one. The question remains whether a 'personal' computer

would need all its features and memory, and it is probably better suited in a business atmosphere.

The Apple III has a built-in disk drive that is much quieter than the old Apple drives. We have recently seen some new drives for the II that are much quieter and more professional sounding. They all seem to work quite reliably however. It also has an RS-232 serial port, two ports that support two analog and two digital devices, B & W and color video ports and an external speaker plug. The III runs on SOS (Sophisticated Operating System), one of the two biggest improvements over the II. It is a system reminiscent of the II's Pascal environment (which it is written in), including its exasperating file names, but it is much better than DOS and CP/M.

There is a standardized interface for all block and character devices, and each device is named and configurable. There is a Utility Configuration program to set up initial parameters and to save your personal configuration to disk. Then, assuming you don't buy any additional hardware, you transfer that file to all your disks that you want to boot and you are all set. We added a Vista Timecard III to our system in about ten minutes, faster than it took us to learn how to set the right time after it was connected. The Timecard is accurate to several seconds per week.

The file system uses pathnames and supports multiple heirarchies in the directory system, a branching tree file structure. Fifty-one files can be entered in a volume file directory, but there can be 1663 sub-directory entries on that volume. File names can be 15 characters long, something we never thought much about on the II, but having worked more on a Flex system allowing only eight characters per name, we really appreciate additional file name lengths. Pathnames can be up to 128 characters.

The other major improvement is in the console, CRT display. Not only does the III standardly display 80 characters per line, upper and lower case (the biggest defect in the II), but also the character set is in RAM, leading to user-definable character sets. In fact, Apple supplies several fonts on the Apple Writer III and Business BASIC disks. We found it fascinating to write an example in Gothic font. While we can't think of any real use for a Gothic font, we could definately see some uses for math symbols and Greek letters. The keyboard also supports type-ahead (no use to someone who types as slow as we do) 128 character codes directly generated from the keyboard (256 if you count the 'open apple' which sets the high bit of all keys on the keyboard), a numeric keypad, and all keys repeatable if held for more than 1/2 second. We especially like the two-speed cursor control keys, which go faster if you push down harder.

The Monitor III that came with the system is the best green monitor that we've seen — crisp and clear. However, the idea that 'normal' is black letters on a green screen and 'inverse' is green letters on a black screen must have been thought up by someone not connected with reality. Actually, in deference to us computer types, the default condition is 'inverse'.

The Apple III appears to us to be a smooth working machine, but too expensive to be a home computer. It is more appropriate than the Apple II for business use.

So what about the IIe? The reviews and our personal observation point out that the structural changes, while major in fact, are minor in operation.

All the products reviewed lived up to our expectations; some can make your Apple do things you wouldn't have dreamed possible. If you want increased performance or to learn about new microprocessors, you don't have to buy a whole new computer.

Manufacturers' Addresses

Excell-9
Norell Data Systems
3400 Wilshire Blvd.
P.O. Box 70127
Los Angeles, CA 90010

The Mill
Stellation Two
P.O. Box 2342
Santa Barbara, CA 93120

Dtack Grounded
Digital Acoustics
1451 E. McFadden, Suite F
Santa Ana, CA 92705

ASSEM68K
Phase Zero, Ltd.
2509 N. Campbell Ave. Suite 130
Tucson, AZ 85719

Apple III
Apple Computer Co., Inc.
Cupertino, CA 95014

The Apple IIe

So what about the IIe? We can hear the questions now. Also, we get telephone calls several times a week asking what are the differences between the II+ and the IIe. Unfortunately, we have not received a product for review. The most important things to determine, such as the changes to the F8 ROM, will have to wait until we can do some experimentation or until some of our readers who own a IIe write in to let us know the differences.

The reviews and our personal observation point out that the structural changes, while major in fact, are minor in operation. To someone who uses a II+ with all the trimmings (a language card, an 80-column board, etc.), the changes will seem minor indeed. The keyboard layout is similar to the III and we never did get comfortable with that. Not having the double quote over the '2' really slowed down my already snail's pace typing rate. We did like the cursor keys, however.

Most software seems to run OK; the original figure of 85% appears to have been a major understatement designed to shock the software community. Unless a routine uses strange entries into the Monitor ROM, the programs should run OK. Eight-column boards are strange creatures, being completely incompatible with each other, so don't expect all programs for an 80-column board to work on the IIe. They wouldn't run on a II+ without the specifically designed board, either.

We promise a more technical in-depth report on the exact changes when we are able to get that information. If anyone has a IIe, write us about the changes. We know that the binary-language loader is certainly faster than the old system master 'HELLO'.

(Continued on next page)

Listing 1: Flex Example (Assembly)

```
FLEX ASSEMBLY LANGUACE PROGRAM : NAMEDISK FOR FLEX ON THE APPLE

                              *  DISK RENAME/RENUMBER
                              *
                              *  COPYRIGHT (C) 1982
                              *  THE COMPUTERIST, INC.
                              *  34 CHELMSFORD STREET
                              *  CHELMSFORD, MA 01824
                              *  ALL RIGHTS RESERVED

                              *  USER-CALLABLE ROUTINES

               CC0C  WKDRV   EQU   $CC0C     WORK DRIVE LOCATION
               CC14  LINBUF  EQU   $CC14     LINE 8UFFER POINTER
               CD03  WARMS   EQU   $CD03     DOS WARMSTART ENTRY
               CD15  GETCHR  EQU   $CD15     GET CHARACTER
               CD18  PUTCHR  EQU   $CD1B     PUT CHARACTER
               CD1B  INBUF   EQU   $CD1B     INPUT 8UFFER LINE
               CD1E  PSTRNG  EQU   $CD1E     PRINT STRING
               CD27  NXTCH   EQU   $CD27     GET NEXT 8UFFER CHARACTER
               CD39  OUTDEC  EQU   $CD39     OUTPUT DECIMAL NUMBER
               CD3F  RPTERR  EQU   $CD3F     REPORT DISK ERROR
               CD4B  INDEC   EQU   $CD4B     INPUT DECIMAL NUMBER

                     *  FMS EQUATES

               D403  FMSCLS  EQU   $D403     FMS CLOSE
               D406  FMS     EQU   $D406     FMS CALL

                     *  SYSTEM EQUATES

               C840  FC8     EQU   $C840     SYSTEM FC8
               C841  ERLOC   EQU   FC8+1     ERROR NUMBER
               CB43  FCBDRV  EQU   FCB+3     DRIVE NUMBER

                     *  MAIN PROGRAM

                             ORG   $C100

C100 20   01         CALL    BRA   CALL2
C102 02              VN      FCB   2         VERSION NUMBER
C103 108E CC14       CALL2   LDY   LINBUF    POINT TO NEXT 8UFFER CHARACTER
C107 A6   A4                 LDA   ,Y        GET FIRST CHAR
C109 81   0D                 CMPA  #$D       IS IT A CR?
C108 26   0B                 BNE   SPEC1
C10D B6   CC0C               LDA   WKDRV     GET WORKING DRIVE NUMBER
C110 B7   CB43               STA   FC8DRV    STORE IN FC8
C113 20   15                 BRA   CALL3
C115 B1   30         SPEC1   CMPA  #'0       IS FIRST CHAR A '0'?
C117 27   0C                 BEQ   SPEC2
C119 81   31                 CMPA  #'1       IS CHAR A '1'?
C11B 27   08                 8EQ   SPEC2
C11D 86   15                 LDA   #21       FORCE ILLEGAL FILE SPECS ERROR
C11F B7   CB41               STA   ERLOC     STORE IN FC8
C122 16   00CB               LBRA  ERROR
C125 B0   30         SPEC2   SUBA  #$30      CONVERT ASCII TO BINARY
C127 87   CB43               STA   FCBDRV    STORE DRIVE NUMBER IN FCB
C12A BE   C840       CALL3   LDX   #FCB      POINT TO FCB
C12D 86   10                 LDA   #16       OPEN SYSTEM INFO RECORD
C12F A7   84                 STA   ,X        STORE IN FCB
C131 BD   D406               JSR   FMS       CALL FMS
C134 1026 0085               LBNE  ERROR     CHECK FOR ERRORS
C138 8E   0007               LDX   #7        GET RECORD CODE
C138 A7   B4                 STA   ,X        STORE IN FCB
C13D BD   D406               JSR   FMS       CALL FMS
C140 1026 00A9               LBNE  ERROR     CHECK FOR ERRORS

C144 8E   C23E               LDX   #STRNAM   POINT TO STORAGE AREA
C147 C6   B0                 LDB   #12B
C149 6F   80         CALL4   CLR   ,X+       CLEAR STORAGE
C14B 5A                      DECB
C14C 26   FB                 BNE   CALL4

C14E BE   C1F6               LDX   #CURNMS   CURRENT NAME STRING
C151 BD   CD1E               JSR   PSTRNG    PRINT IT
C154 8E   CB90               LDX   #FCB+80   POINT TO VOLUME NAME
C157 C6   08                 LDB   #11       PREPARE TO OUTPUT 11 CHAR
C159 A6   B0         LOOP1   LDA   ,X+       LOAD DATA, 8UMP ADDRESS
C15B BD   CD18               JSR   PUTCHR    OUTPUT NEXT CHARACTER
C15E 5A                      DECB
C15F 26   F8                 BNE   LOOP1     PRINT UNTIL DONE
C161 8E   C208               LDX   #NEWNMS   NEW NAME STRING
C164 BD   CD1E               JSR   PSTRNG    PRINT IT
C167 8D   CD1B               JSR   INBUF     INITIATE BUFFER INPUT
C16A 8E   C23E               LDX   #STRNAM   POINT TO STORAGE AREA
C16D 8D   CD27       LOOP2   JSR   NXTCH     GET NEXT BUFFER CHARACTER
C170 25   04                 8CS   FILL1     EXIT ON NON-ALPHA CHAR

C172 A7   B0                 STA   ,X+       STORE IN WORK AREA
C174 20   F7                 8RA   LOOP2     TRANSFER UNTIL DONE
C176 8C   C23E       FILL1   CMPX  #STRNAM   WAS FIRST CHAR NON-ALPHA
                                             (CR)?
C179 27   20                 8EQ   NUMBER    IF SO, SKIP TO DISK NUMBER
C17B 86   FF                 LDA   #$FF
C17D A7   B4                 STA   ,X        FLAG END OF NAME
C17F 8E   C23E               LDX   #STRNAM   POINT TO INPUT NAME
C182 10BE C890               LDY   #FC8+80   POINT TO NAME IN FCB
C186 C6   0B                 LD8   #11
C188 A6   80         FILL2   LDA   ,X+       GET CHAR FROM INPUT NAME
C18A 81   FF                 CMPA  #$FF      END OF NAME?
C1BC 27   07                 BEQ   FILL3     IF SO, PAD WITH NULLS
C18E A7   A0                 STA   ,Y+       ELSE, TRANSFER TO FC8
C190 5A                      DECB
C191 26   F5                 8NE   FILL2     TRANSFER UNTIL FLAG
                                             REACHED
C193 20   06                 BRA   NUMBER    NO PADDING NEEDED
C195 4F              FILL3   CLRA
C196 A7   A0                 STA   ,Y+       STORE NULLS IN FCB
C198 5A                      DECB
C199 26   FA                 BNE   FILL3     PAD UNTIL DONE

C19B 8E   C21A       NUMBER  LDX   #CURNMB   CURRENT NUMBER STRING
C19E BD   CD1E               JSR   PSTRNG    PRINT IT
C1A1 BE   CB9B               LDX   #FCB+91   POINT TO VOLUME NUMBER
C1A4 5F                      CLRB            SUPPRESS PADDING
C1A5 BD   CD39               JSR   OUTDEC    OUTPUT DECIMAL NUMBER
C1AB 8E   C22C               LDX   #NEWNMB   NEW NUMBER STRINC
C1AB BD   CD1E               JSR   PSTRNG    PRINT IT
C1AE BD   CD1B               JSR   INBUF     GET LINE FROM INPUT BUFFER
C1B1 108E CC14               LDY   LINBUF    POINT TO FIRST CHARACTER
C1B5 BD   CD48               JSR   INDEC     INPUT DECIMAL NUMBER
C1B8 25   1C                 BCS   WRITE     EXIT ON NON-NUMERIC
                                             CHARACTER
C1BA 5D                      TST8
C1BB 27   19                 8EQ   WRITE     EXIT ON INVALID NUMBER
C1BD 5F                      CLRB
C1BE A6   A0         NUMB2   LDA   ,Y+       GET NEXT DIGIT
C1C0 81   0D                 CMPA  #$D       IS IT A CR?
C1C2 27   03                 BEQ   NUMB3     EXIT IF SO
C1C4 5C                      INCB
C1C5 20   F7                 8RA   NUMB2     GET DIGITS UNTIL CR REACHED
C1C7 C1   04         NUMB3   CMP8  #4
C1C9 22   0B                 8HI   WRITE     ACCEPT IT IF LESS
                                             THAN 10000
C1CB 10BE CB9B               LDY   #FCB+91   POINT TO VOLUME NUMBER
C1CF 4F                      CLRA
C1D0 A7   A4                 STA   ,Y        CLEAR OUT NUMBER BYTES
C1D2 A7   21                 STA   1,Y
C1D4 AF   A4                 STX   ,Y        STORE VOLUME NUMBER IN FC8
C1D6 8E   C840       WRITE   LDX   #FC8      POINT TO FCB
C1D9 6F   88 1E              CLR   30,X      TRACK NUMBER IN FC8
C1DC B6   03                 LDA   #3        SECTOR NUMBER
C1DE A7   88 1F              STA   31,X      STORE IN FCB
C1E1 B6   0A                 LDA   #10       WRITE ONE SECTOR
C1E3 A7   84                 STA   ,X        STORE IN FC8

C1E5 8D   D406               JSR   FMS       CALL FMS
C1E8 26   03                 8NE   ERROR     CHECK FOR ERRORS
C1EA 7E   CD03               JMP   WARMS     RETURN TO FLEX

C1ED BD   CD3F       ERROR   JSR   RPTERR    REPORT ERROR
C1F0 BD   D403               JSR   FMSCLS    CLOSE FMS
C1F3 7E   CD03               JMP   WARMS     RETURN TO FLEX

                     *  STRING EQUATES

C1F6 43 55 52 52     CURNMS  FCC   'CURRENT NAME =    ',$04
C208 4E 45 57 20     NEWNMS  FCC   'NEW NAME?         ',$04
C21A 43 55 52 52     CURNMB  FCC   'CURRENT NUMBER =  ',$04
C22C 4E 45 57 20     NEWNMB  FCC   'NEW NUMBER?       ',$04
                     *  NAME STORAGE AREA

C23E                 STRNAM  RMB   128

                             END   CALL
```

Listing 2: Mill Example (Assembly)

```
                        DEMO PROGRAM FOR THE MILL

                                    *   McMill Demo
                                    *   Feb 25, 1983
                                    *   From Programming the 6809
                                    *   Modified for relative addressing

                                         ORG    $7FFE
                        7FFE 0000        FCB    $00,$00
                        8000 308D0028 BUBBLE LEAX  BASE,PCR  GET TABLE
                        8004 C62C        LDB    #LENGTH
                        8006 5A          DECB
                        8007 3085        LEAX   B,X       Point to end
                        8009 6F8D001E    CLR    EXCHG,PCR Clear exchange flag
                        800D A684   NEXT LDA    ,X        A=Current entry
                        800F A182        CMPA   ,-X       Compare with next
                        8011 2C0E        BGE    NOSWIT    Go to Noswit if curr.&=next
                        8013 3404        PSHS   B         Save B
                        8015 E684        LDB    ,X        Get next
                        8017 E701        STB    1,X       Store in current
                        8019 A784        STA    ,X        Store current in next
                        801B 3504        PULS   B         Restore B
                        801D 6C8D000A    INC    EXCHG,PCR Set exchange flag
                        8021 5A   NOSWIT DECB             Decrement B
                        8022 26E9        BNE    NEXT      Continue until 0
                        8024 6D8D0003    TST    EXCHG,PCR Exchanged=0?
                        8028 26D6        BNE    BUBBLE    Restart if not 0
                        802A 39          RTS
                        802B       EXCHG RMB    1
                                   BASE  EQU    *
                        802C 1A   LENGTH FCB    26
                        802D 4744525548  FCC    "GDRUHBSEQIGNVC"
                        8033 4253455149474E56
                        803B 43
                        803B 4B4654554F  FCC    "KFTUOPLFDSQC"
                        8041 504C4644535143
```

Listing 3: 68000 Example (Assembly)

```
68000 ASSEMBLY LANGUAGE PROGRAM : STRING SEARCH

                               ORG      $2000
002000: 207C00002040           MOVEA.L #MAIN,A0      ;LOW RANGE LIMIT
002006: 227C00002053           MOVEA.L #TEST,A1      ;HIGH RANGE LIMIT
00200C: 7003                   MOVE.L  #3,D0         ;TEST STRING LENGTH
00200E: 48A7000A               MOVEM   D1/D3,-(SP)   ;SAVE REGISTERS
002012: 48E70900               MOVEM.L A0/A3,-(SP)   ;ON STACK
                     *
                     * SEARCH FOR FIRST CHAR OF TEST STRING
                     *
002016: 1611                   MOVE.B  (A1),D3       ;FIRST TEST CHAR
002018: 95CA         FIRST     SUBA.L  A2,A2         ;CLEAR A2 TO START
00201A: 0C10002A     CHKEND    CMPI.B  #'*',(A0)     ;END OF MAIN?
00201E: 671E                   BEQ.S   RETRN         ;YES. GO EXIT
002020: B618                   CMP.B   (A0)+,D3      ;FIRST CHAR MATCH?
002022: 66F6                   BNE.S   CHKEND        ;NO. KEEP SEARCHING
                     *
                     * FIRST TEST CHAR FOUND, COMPARE REST
                     *
002024: 3200                   MOVE    D0,D1         ;STORE TEST LENGTH
002026: 5541                   SUBQ    #2,D1         ;D1=COUNT-2
002028: 2649                   MOVEA.L A1,A3         ;STORE TEST ADDR.
00202A: 528B                   ADDQ.L  #1,A3         ;LOOK AT CHAR TWO
00202C: 2448                   MOVEA.L A0,A2         ;STORE MAIN ADDR.
00202E: 538A                   SUBQ.L  #1,A2
002030: 0C10002A     LOOP      CMPI.B  #'*',(A0)     ;END OF MAIN?
002034: 6708                   BEQ.S   RETRN         ;IF SO, RETURN
002036: B10B                   CMPM.B  (A3)+,(A0)+   ;COMPARE CHARS
002038: 66DE                   BNE.S   FIRST         ;MATCH FAILED
00203A: 51C9FFF4               DBF     D1,LOOP       ;KEEP COMPARING
00203E: 4E4F         RETRN     TRAP    #15           ;DUMP REGS, EXIT
002040: 57415348494E MAIN      ASC     'WASHINGTON BULLETS*'
002053: 544F4E       TEST      ASC     'TON'
```

Listing 4: 68000 Example (BASIC)

```
APPLESOFT PROGRAM : UPLOADS MACHINE LANGUAGE PROGRAM
ONTO 68000 BOARD AND EXECUTES IT.

10 A = PEEK (49280): CALL  - 936
20  PRINT CHR$ (4);"BLOAD UTIL2,A$8600,D1"
    :CALL 38383
30  PRINT CHR$ (4);"BLOAD SEARCH.OBJ0,D2,A$6000"
40 P = 38383:Q = P - 102: CALL 38380
50  CALL Q: REM 07 002000 6008 0056
60  CALL Q: REM 02 002000
70  CALL P - 27
80  PRINT
    :PRINT "IF THE TEST STRING WAS FOUND, REGISTER"
90  PRINT "A2 WILL BE  00 00 20 47."
100  PRINT
    :PRINT "IF THE TEST STRING WAS NOT FOUND,"
110  PRINT "REGISTER A2 WILL BE  00 00 00 00120  END
```

MICRO

Transition Period

Commodore's product line has evolved quickly over the 4½ years since the first PETs were delivered. The heart of the PET is the 6502 microprocessor, designed by Commodore's (then) newly acquired subsidiary, MOS Technology. At that time the company's only computer product was the small-keyboard PET, long since phased out in favor of models with full-sized business and graphics keyboards. From these evolved an 80-column model, which later gave rise to the 96K CBM 8096 and the SuperPET. With the exception of the SuperPET (or the 6809 half of it), all are essentially the same computer.

Then Commodore introduced the VIC-20, billed as the "first full-featured computer for under $300." It was named after its technological centerpiece, an inexpensive CRT controller called VIC (Video Interface Chip). Many things about the VIC-20 are radically different. Some are obvious compromises to obtain a low price: the keyboard has fewer keys and no numeric keypad; the screen consists of only 23 rows of 22 characters; only 5K of RAM is included; a separate TV or monitor is required; the IEEE-488 bus is gone; and no cassette player is included. There are many improvements aimed particularly at the home market (as opposed to the hobbyist market of the PET): color, sound, a controller port, programmable function keys, and program cartridge capability. The VIC, even with all its differences, is easily identifiable as a descendant of the PET. The operating system is essentially the same as the PET and CBM; PET/CBM BASIC programs run on a VIC with relatively few adjustments; and the processor is still a 6502.

Commodore New Wave

by Loren Wright

— Commodore 64 — First in New Wave

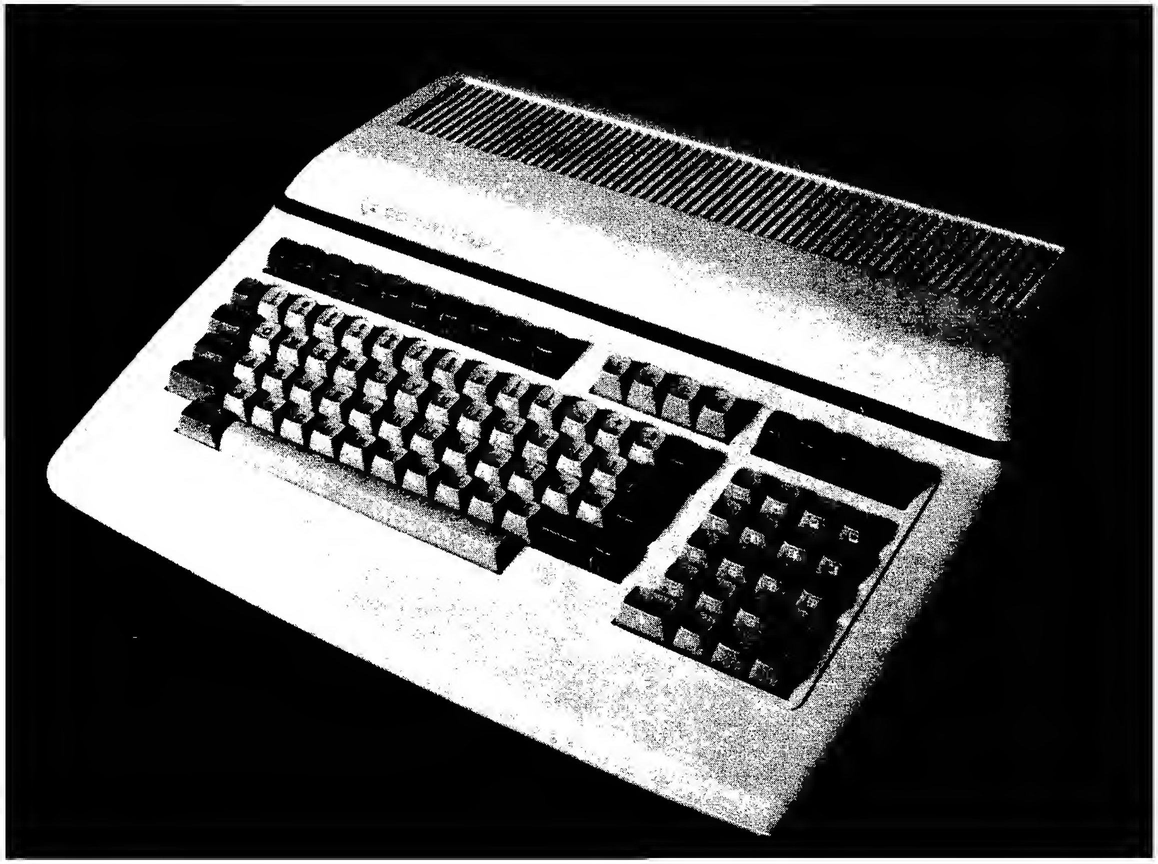
At the same time that Commodore was gearing up for a massive VIC-20 advertising campaign last fall, it introduced the Commodore 64. The Commodore 64 looks very much like the VIC-20 (it even appears that the same mold was used for the case), but the inside is a lot different. Featured are four new chips: the 6510 microprocessor, the 6526 CIA interface chip, the 6581 SID sound chip, and 6567 VIC II video chip.

The 6510 is a 6502 with an added I/O port, which is used to switch RAM, ROM, and I/O in and out of the 64K address space. There is actually 64K of RAM in there that could be used all at once, if you wanted to design your own operating system. The practical advantage is that things like the character generator and I/O chips can share the same address space and get switched in only when needed. Under the BASIC operating system provided, this means that about 8K more RAM is available for program storage. As in the VIC, the most fundamental tasks, such as getting a character from the keyboard and putting a character on the screen, are concentrated in a single 4K ROM called the KERNAL. This design feature, combined with the built-in switching, makes the C64 ideal for such languages as Pascal and FORTH, which have much of their own operating systems built in. When the BASIC ROMs permanently occupy address space as they do in the PET, these languages use additional space, leaving less for programs, or operate from the disk, making operation slow. With the C64, the BASIC ROMs can be switched out and RAM (or cartridge ROM) can be switched into the same space.

The VIC II is a considerable improvement over the VIC. It allows 25 rows of 40 characters, fine scrolling, high-resolution graphics, and eight hardware-maintained movable object blocks (called sprites). With the relative abundance of RAM, high-resolution graphics and user-defined character sets are more practical than they are with the limited memory of the VIC-20. For detailed information on the capabilities of the VIC II chip, see my December 1982 PET Vet column (MICRO 55:54).

The SID is an extremely powerful sound chip with four different waveforms, envelope generators, filters, ring modulation, and synchronization for each of three voices. See my February 1983 PET Vet column (MICRO 57:71) for more on the synthesizer-like capabilities of the SID chip.

The CIA adds a time-of-day clock to the familiar 6522 VIA. Two of them are



Commodore's C 128/80

used in the C64 — one for the keyboard and one for the parallel user port.

The C64 apparently will give rise to a whole line of portable computers designed to compete favorably with the Osborne portable computers. The unit is smaller than the Osborne, and it will be available in three different models: 1. a 5-inch black and white monitor and single 5¼-inch floppy drive, 2. a 5-inch color monitor and single floppy drive, and 3. a color monitor and dual floppy drive. The third will sell for less than $1700; the others less.

This line was previewed in prototype versions at the Consumer Electronics Show in Las Vegas in January. The machines are apparently a C64 complete with 6510, SID, VIC II, and CIA's, but in a different configuration. Further details were unavailable from Commodore, and there is no telling exactly when these machines will appear in stores.

6509 Addresses 1 Megabyte

Three new models, all based on another new processor (the 6509) were announced last year. They have received various names (500 and 700 series; PET II and CMB II; B, P, and BX series), but now it seems that only one, the C 128/80, will appear soon.

Like the C64 and VIC-20, the keyboard and computer are combined in a single low-profile unit. When this unit is connected to a black and white monitor, it produces an 80-column display of 9x14 characters. Both business and PET-graphics character sets are included. The keyboard is expanded considerably. There is a separate numeric keyboard with double-zero, 'clear-entry', and double-size 'enter' keys. There is a row of 10 programmable function keys across the top of the main keyboard. The cursor controls are on four separate keys, rather than only two. Also, unlike the VIC and the C64, the unit includes a switching power supply and 'whisper' fan.

As its name implies, the C 128/80 includes 128K of RAM, and that is expandable internally to 256K and externally to 896K. This is accomplished with the new 6509 microprocessor, which performs its own bank switching to address this much memory. The instruction set is identical to that of the 6502.

The BASIC supplied in 24K of ROM is a further advancement beyond BASIC 4.0 (5.0?). Apparently programs written in BASIC 4.0 will run under the new BASIC.

The C 128/80 has a number of different interfaces. The IEEE-488 interface makes it possible to use any Commodore peripheral, including floppy and hard disk drives and several printers. The IEEE-488 bus is also used by Hewlett-Packard, Tektronix, and Fluke in many of their scientific instruments. The RS-232C interface, accomplished with a 6551 ACIA chip, makes a great number of peripherals available including auto-answer/auto-dial modems and inexpensive printers. Also available is the parallel user port supported by the 6526 CIA chip.

The 6581 sound chip, with all the sound capabilities, is also included.

Breakers on the Horizon

Commodore's Max machine (or Ultimax?) has not appeared yet, perhaps because of the current success of the VIC-20. Also, I presume there will be machines in the same line as the C 128/80, only with different features (a color monitor, included disk drives?).

Commodore showed a prototype of a hand-held computer, but sources indicate that product won't see the light of day.

You may contact Loren Wright at P.O. Box 6502, Amherst, NH 03031.

MICRO

IS 16-BITS THE SOLUTION ?

by Keith Roberson

Sixteen-bit microprocessors have made a significant impact on the imagination of microcomputer users, for a good reason. But it's just possible that for the novice small business user the distinction is moot, if not actually irrelevant. This article discusses why 8-bit technology, while far from the leading edge, can be expected to have a long and active life in the small computer marketplace.

The 8-bit microprocessor has come a long way since its inception; from a hobbyist's toy to the foundation of the burgeoning business computer marketplace. The inroduction of a new microprocessor generation — the high peformance 16-bit systems — marks another step toward maturity. This event has moved some industry observers to see in it an abrupt end to the product life of the eight-bit business computer system.

The new computers do, in fact, have certain advantages over the older technology. Faster and more powerful than their 8-bit predecessors, they are already making 8-bit technology obsolete.

Or are they?

Ask a potential small business computer buyer for an opinion on the subject. Chances are he won't know what you're talking about — or care.

And why should he?

Most microcomputer business systems buyers are technologically uninformed. Many prefer to keep it that way. Trend-setters in their own right, they accept the microcomputer as a potentially useful tool for solving their business and information management problems. Accountants, clerical workers, executives, or professionals, few have used anything more complex than an adding machine or a hand-held calculator before. They want economical, efficient, and productive solutions to their business problems. The details of a technology that achieves this could hardly interest them less.

They must deal with many factors and are confronted by an incredible number of factors in making a choice among business computers for their own applications.

Asking them to choose between 8-bit and 16-bit microprocessor technology is to add to the complexity of an already enormously confusing mass of variables. Since any such decision in the last analysis must be based on the harsh realities of business, the objective of the selection process is to choose the computer that does the best job in the specific installation, at the lowest cost.

The issue of 8-bit *versus* 16-bit business computers must be secondary to this concern. Nevertheless, the 16-bit computer system is drawing vast amounts of attention to itself. To an uninformed observer, the claims carry a measure of validity. Some discussion of these claims, and a comparison of eight-bit and 16-bit systems may help clarify the issues for prospective business computer buyers.

Bits, Bytes, and MIPS

Sixteen-bit systems are often described as high performance machines. That there are major differences between 8- and 16-bit microprocessors cannot be disputed. However, that does not mean 16-bit technology is the fix-all-problems solution to business microcomputer problems. Many factors can affect overall system performance, and the most ad-

vanced technology is not always the most appropriate.

Comparisons of the two technologies are usually made on the basis of performance. But software availability, expansion capability, and price are factors a buyer must definitely consider in the process of selecting a high-ticket capital item like a computer that will be used for several years. And performance itself must be viewed within the parameters of realistic settings and the constraints these impose on a system.

Performance

Perhaps the most widely used — and most misleading — measure of a computer's performance is the number of binary digits (bits) the Central Processing Unit (CPU) can access or manipulate at one time. Bits constitute the atoms of a computer. They are processed in clusters (words) of 8, 16, 32, 64, or more at one time. The larger the word, the more information a computer can process in a given time. Hence the concern over word size.

A 16-bit processor such as the Intel 8086 has twice the word size of Zilog's 8-bit Z80, but that does not mean it has twice the performance. Sixteen-bit microprocessor chips come in many forms and configurations. The Intel 8088, for example, processes a 16-bit word internally, but transfers it *via* an 8-bit external data path (bus), in order to allow designers of 16-bit systems to incorporate it into existing 8-bit designs. On the other hand, Motorola's 68000 has a 32-bit internal architecture and a 16-bit bus. yet both are considered 16-bit microprocessors.

The performance of these wider bandwidth computers is nearly always limited by their application. The application itself can impose conditions that nullify any performance advantage provided by a state of the art technology. For example, word processing — which accounts for the bulk of current small business microcomputer applications — involves the extensive manipulation of alphanumeric data. But alphanumeric data is represented in an 8-bit format. In such an application, the capabilities of a 16-bit microprocessor are grossly underutilized. The buyer ends up paying for excessive unused capability. However, extensive scientific calculations — such as Fourier transforms, for example — could show dramatic increases in through-put by utilizing the wider internal bandwidth of 16-bit processors.

In these popular configurations, the critical path to high performance is not RAM size but speed of the disk storage unit. This is not at all affected by the CPU, whether it be 8-bit or 16-bit.

MIPs and Clock Speed

A better performance benchmark than word size, or the number of bits in a system, is the number of instructions or operations a processor can execute in a given amount of time.

The unit of measure generally used is MIPS (million instructions per second). This value is largely dependent on the CPU clock speed. Typically, the new 16-bit processors have clock speeds about 50% faster than 8-bit chips, and have bus widths that in many instances allow them to handle twice as much data at a time.

Faster clock speeds account for most of the perceived performance increases of 16-bit microprocessors over their 8-bit predecessors. The Intel 8086, for example, runs at 6 MHz. Taking into account its word size, and a slightly more powerful instruction set, it could be said to run twice as fast as an 8-bit Z80A running at 4 MHz. However, in many applications the 8086 is comparable in performance to a Z80B running at 6 MHz, and slower than a Z80H running at 8 MHz.

Internal and External Memory

One of the indisputable advantages of a 16-bit system is its inherent ability to address or access more random access memory (RAM or main memory), typically a megabyte or more, compared to the 64-kilobyte limit of the 8-bit chips. However, advances made through a technique called bank switching, allow 8-bit micros to access many times their nominal 64K limit.

A large internal memory makes it possible to design a comuter that can use a sophisticated operating system (OS) — a program always resident in RAM that oversees the loading and execution of the application programs. A large RAM also reduces the need for frequent access to the much slower disk memory by keeping larger portions of program and data in the much faster RAM.

But RAM is one of the most expensive parts of a computer system. And a megabyte of internal memory is far in excess of what today's most popular applications programs are designed to use.

Most business application programs available today are disk intensive. That is, they rely heavily on disk media for external memory or data storage. In these popular configurations, the critical path to high performance is not RAM size but speed of the disk storage unit. This is not at all affected by the CPU, whether it be 8-bit or 16-bit.

There are two main types of disks and disk drives: hard and floppy. A hard disk costs about five times as much as a floppy, is orders of magnitude faster, and offers on-line storage accessibility measured in megabytes, as opposed to the kilobytes of storage offered by floppies.

Two other features of fast disks that are not remarkably affected by the use of 16-bit microprocessors are direct memory access (DMA) — which optimizes the transfer time from disk to memory, and buffering, the use of a temporary memory as a holding area to reduce frequent disk access.

Software

Most often considered a separate issue, software is intimately associated with performace. The finest hardware is useless without strong applications and systems software. What sometimes appears to be a troublesome performance problem is often resolved through the use of redesigned software.

Eight-bit based systems enjoy a distinct advantage here, especially when coupled with Digital Research's CP/M operating system (or a compatible variation of it), the industry's de-

facto operating system. Thousands of CP/M applications exist for 8-bit systems, written and tested over the many years since microcomputers emerged as a valid tool for use in business and other settings. Compared to the few hundred applications presently available for 16-bit machines, the advantage of implementing an 8-bit system is clear, where the objective is to be up and running with the largest selection of application software from which to select suitable programs.

Another factor to consider in evaluating the appropriateness of 16-bit over 8-bit installations, is the fact that the bulk of 16-bit programs are derived from direct analogs of the 8-bit versions and do not really take advantage of the 16-bit's potential for increased performance. While this may not be an important concern for users who plan to develop their own applications programs, it can be a significant consideration for those whose objective is to use the new machine as soon as possible to solve pressing business or professional problems.

Interestingly enough, many of the most talked about recent advances in applications software (such as multiuser and shared data base programs) were developed on 8-bit machines and then ported over to the 16's. This is not quite so strange as it sounds; most 16-bit machine manufacturers still rely on 8-bit computer sales for their bread and butter.

Expandability and Multi-user Capability

The ability to expand — whether it be the size of internal or external memory, or the number of users it will support — is a frequently over-looked but important measure of performance.

The decision on which system to obtain must be based as much on current needs as on the needs of the predictable future.

What if more users need to access the system? What will happen to system performance if more users are added? Is the current storage capacity adequate? If not, can the storage capacity be expanded? Will it be possible to modify the system readily, or must it be scrapped and another system selected to replace it?

In short, is it a system with a reasonable productive life in the setting where it will be used?

For most serious business computer users, multi-user capability is a must. As a business grows, more ways to use a computer are soon discovered. Within a surprisingly short time, it becomes evident that a single-user system — be it a 16-bit business computer or an 8-bit personal computer enhanced to its limits to fit into such a setting — is inadequate.

Along with the increased number of applications, the need for easy access to common files and the most recent information also grows.

If anticipated computer usage for the future justifies selecting a system that can easily accommodate additional users and memory expansion, at least two major options are currently available: additional single-user workstations or multi-user systems.

Multi-user systems are available in two architectures: *time sharing* and *distributed processing.*

Time-Sharing

Time-sharing allows several users to share a single computer apparently simultaneously. The system behaves as if the computer serves only that one user, when, in fact, the user controls only one time slice of many. Time-sharing has its origins in traditional mainframe and minicomputer operations, and is based on the most economical utilization of expensive resources.

A user program takes control of a CPU, does some processing for a while, then gives up the CPU to another user.

The immediate cost advantage of time-sharing is the ability to add more users without significant hardware expenditures. However, performance rapidly deteriorates as more users are added. Even 16-bit microprocessors show significant degradation, sometimes slowing down to unacceptable response rates with the addition of only one or two users.

Additional overhead is also paid in performance and memory space for the operating system to handle such necessary functions as memory management, scheduling and device polling.

Distributed processing

In distributed processing, several microcomputers are linked together to perform like one big computer. users may be added without significant degradation of system performance.

The costs associated with extensive cabling, stand-alone computer installations and their maintenance, must be evaluated against the advantages of this approach. Cost effectiveness is always an important consideration in any decision involving a business.

An interesting compromise to both types of systems has been made by a number of systems that utilize a dual processor architecture. users share 8-and 16-bit processors transparently; indeed, users may run 8-bit and 16-bit on the same system concurrently. This provides the opportunity to use 8-bit software available today, and offers a viable migration path to the 16-bit as more sophisticated software for it becomes available. However, true parallel operation is not generally possible on these systems. Most of them use tightly-coupled processors, and, in most cases, only one processor is running at a time. This reduces the cost/benefit factor to the user by providing only very slight performance advantage over a single CPU system.

An easy-to-use 16-bit operating system may be hampered by a lack of software.... Even 16-bit microprocessors show significant degradation sometimes slowing down to unacceptable response rates....

All things considered, buyers of business computer systems may need to pay more attention to future expansion capability than processing speed when evaluating the performance benefits of 8-bit *versus* 16-bit computers.

Price Considerations

If price were the only consideration, 8-bit machines would win hands down, since they make up virtually the entire low-priced end of the market. However, price must be coupled with performance; an Apple may be bought

for under $2000, but could cost twice as much in additional hardware to perform the most simple business functions at reasonable levels. Nor will it ever be able to do the things a true business computer can do. By the same token, an IBM mainframe could do it all, but the cost would not be justified.

Picking a system with the best price performance ratio is a matter of deciding which features are needed and finding the lowest-priced system with those features.

Other Considerations

Possibly the worst obstacle to more rapid integration of microcomputers into the society can be found in the industry's lack of standardization. Incompatible and untried operating systems abound. Users often are unwittingly obliged to serve as a test site.

Among multiuser systems, some operating systems may be promising at this point of their development. It remains for the individual user to evaluate the benefits of any operating system, whether it be MS/DOS, CP/M86, UNIX, or a UNIX look-alike, running in a 16-bit environment, over a CP/M, or CP/M compatible, operating system running in an 8-bit environment. Each approach has it benefits, along with its disadvantages. An easy-to-use 16-bit operating system may be hampered by a lack of software. A multiuser MP/M operating system may be forcing the 8-bit microprocessor to function at the extremes of its capability, leaving it vulnerable to malfunctions.

A recently developed approach has been adopted by several manufacturers, who have designed multi-user networked operating systems that are compatible with CP/M. These include MUSE, CP/NET, MMOST, NSTAR and several others.

Given the need for caution in making capital investments, the probability remains that eight-bit operating systems with CP/M capability, will continue to remain the preferred choice of business computer users for some time to come.

The Solution

Today's small-business computer users are looking for ways to handle their work more efficiently. The 16-bit systems are appropriate in some applications, but in many more the best choice may depend more on established 8-bit technology.

There is no question that the bigger processors will eventually replace the 8-bit processors. The current lack of 16-bit software, however, and the predictable need to expand capacity, combine to make high performance distributed processing systems a viable alternative while waiting for the microcomputer to mature more fully. Indeed, some industry analysts think that the new 32-bit processors will take over before the 16-bits have a chance to take hold, and most are now in agreement that the 8-bit will still be around for many years to come.

Z80, Z80A, Z80B, Z80H are registered trademarks of Zilog. **8086, 8088** are registered trademarks of Intel Corporation. **CP/M** and **MP/M** are registered trademarks of Digital Research Corporation.® **UNIX** is a trademark of Bell Laboratories. **Zeus4** is a registered trademark of OSM Corporation.

You may contact Keith Roberson at OSM Computer Corporation, Mountain View, CA.

MICRO

Alspa Computer, Inc.

The price-performance leader. Includes Z80A, 1 or 2 full 8" drives (double density, double sided), 3 serial and 1 parallel port, and winchester port. Prices start at less than $2000. DEALER and OEM inquiries invited.

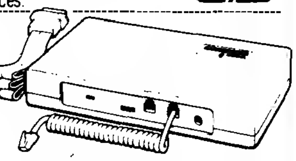
SPECIALS on INTREGATED CIRCUITS

6502	7.45	10/ 6.95	50/ 6.55	100/ 6.15
6502A/6512A	8.40	10/ 7.95	50/ 7.35	100/ 6.90
6520 PIA	5.15	10/ 4.90	50/ 4.45	100/ 4.15
6522 VIA	6.45	10/ 6.10	50/ 5.75	100/ 5.45
6532	7.90	10/ 7.40	50/ 7.00	100/ 6.60
2114-L200		2.45	25/ 2.30	100/ 2.15
2716 EPROM		4.90	5/ 4.50	10/ 4.00
2532 EPROM		7.90	5/ 7.45	10/ 6.90
6116 2KX8 CMOS RAM		7.90	5/ 7.45	10/ 6.90
4116 RAM		8 for 14		
Zero Insertion Force 24 pin Socket (Scanbe)				2.00

Hewlett Packard

Write or call for prices.

Anchor Automation Signalman Modems

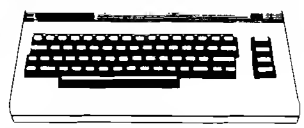


FREE SOURCE MEMBERSHIP WITH SIGNALMAN

All Signalman Modems are Direct Connect, and include cables to connect to your computer and to the telephone. Signalman Modems provide the best price-performance values, and start at less than $100. **Dealer and OEM inquiries invited**

Mark I RS232	(99)	79
Mark II for Atari 850	(99)	79
Mark IV for CBM/PET with software	(169)	119
Mark V for Osborne (software available)	(129)	93
Mark VI for IBM Personal Computer	(279)	195
Mark VII Auto Dial/Auto Answer	(179)	119
Mark VIII Bell 212 Auto Dial/Answer	(399)	319

DC HAYES Smartmodem 219
DC Hayes Smartmodem 1200 545



Apple Emulator for Commodore 64	89
Screenmaker 80 COLUMN CARD for C64	155
Solid Oak 2 Level Stand for C64 or VIC	29
C64/VIC Switch (networking)	135
BACKUP V1.0 tape copier for C64 or VIC	20
CARDBOARD/6 Motherboard - VIC	79
CARDAPTER/1 Atari VCS Adapter - VIC	69
CARDPRINT Printer Interface - C64/VIC	64
CARDBOARD/3s Motherboard - VIC	32
CARDETTE/1 Cassette Interface - C64/VIC	32
CARDRITER Lightpen - C64/VIC	32
CARDRAM/16 RAM Expansion - VIC	64

We carry Apple II+ from Bell & Howell

Apple Emulator for Commodore 64	89
16K RAM Card for Apple	59
Solid Oak 2 Level Stand for Apple	29
Super Serial Card	149
Thunderclock Plus	119
Z80 Softcard and CP/M (Microsoft)	235
Parallel Printer Interface/Cable	79
Apple Dumpling (Microtek) Printer Interface	115
Apple Dumpling with 16K Buffer	160
Grappler + Interface	140
TG Products for Apple in stock	
DC Hayes Micromodem II	299
Videx 80 Column Card	239
Hayden Software for Apple 20% OFF	
PIE Writer Word Processor	120

commodore

See us for Personal, Business, and Educational requirements. Educational Discounts available.

PETSCAN $245 base price

Allows you to connect up to 30 CBM/PET Computers to shared disk drives and printers. Completely transparent to the user. Perfect for schools or multiple word processing configurations. Base configuration supports 2 computers. Additional computer hookups $100 each.

Commodore COMMUNICATES!

COMPACK $115

Intelligent Terminal Package includes: ACIA hardware based interface; DB25 Cable and STCP Software with remote telemetry, transfer to/from disk, printer output, XON-XOFF control, user program control, status line.

VE-2 IEEE to Parallel Interface 110

Includes case, power supply, full 8-bit transmission, and switch selectable character conversion to ASCII

VIC 20 Products		VIC Sargon II Chess	32
BACKUP V1.0	20	**VIC GORF**	32
VIC RAM Cards in stock		**Meteor Run (UMI)**	39
VIC SuperExpander	53	**VIC Radar Ratrace**	24
VIC 16K RAM	69	**Amok (UMI)**	20
Thorn EMI Software		Snakman	15
HES Software		Rubik's Cube	13
VIC Omega Race	32	Programmers Reference	15
Spiders of Mars (UMI)	39	Renaissance (UMI)	39
Programmers Aid	45	VIC Adventure Series	

VICTORY Software for VIC and C64

Street Sweepers	12	Maze in 3-D	12
Night Rider	11	Cosmic Debris	12
Treasures of Bat Cave	12	Grave Robbers Advent.	11
Games Pack I	12	Games Pack II	12
Victory Casino	8	Adventure Pack I	12
Adventure Pack II	12	Trek	11

Commodore 64 Programmers Reference Guide	17
Compute!'s First Book of PET/CBM	11
POWER ROM Utilities for PET/CBM	78
WordPro 3+ - 32K CBM, disk, printer	195
WordPro 3+/64	69
WordPro 4+ - 8032, disk, printer	300
SPELLMASTER spelling checker for WordPro	170
VISICALC for PET, ATARI, or Apple	190
PETRAX PET to Epson Graphics Software	40
SM-KIT enhanced PET/CBM ROM Utilities	35
Programmers Toolkit - PET ROM Utilities	35
Branding Iron EPROM Programmer for CBM	75
PET Spacemaker II ROM Switch	36
2 Meter PET to IEEE or IEEE to IEEE Cable	40
Dust Cover for PET, CBM, 4040, or 8050	8
VIC or C64 Parallel Printer Interface	64
CmC IEEE-RS232 Printer Interface — PET	120
SADI Intelligent IEEE-RS232 or parallel	235
ZRAM - CBM 64K RAM, Z80, CP/M	550
Programming the PET/CBM (Compute!) — R. West	20
Compute! First Book of VIC	11
Whole PET Catalog (Midnight Gazette)	8
Color Chart Video Board for PET	125
PET Fun and Games (Cursor)	11
Disk ICU—Disk Recovery System for PET/CBM	40
Copy-Writer Word Processor for C64, PET/CBM	70

FlexFile for PET CBM, C64 $110

Database, Report Writer with Calculations, Mailing Lists

FORTH for PET full FIG model — Cargill/Riley	$50
Metacompiler for FORTH for independent object code	30
KMMM PASCAL for PET/CBM/C64	85
EARL for PET/CBM Disk-based ASSEMBLER	65
Super Graphics — BASIC Language Exercises	45
Fast machine language graphics routines for PET/CBM	
RAM/ROM for PET/CBM	4K $75 8K $90

DISK SPECIALS

Scotch (3M) 5" ss/dd	10/ 2.25	50/ 2.10	100/ 2.05
Scotch (3M) 5" ds/dd	10/ 3.15	50/ 2.90	100/ 2.85
Scotch (3M) 8" ss/sd	10/ 2.40	50/ 2.20	100/ 2.15
Scotch (3M) 8" ss/dd	10/ 2.95	50/ 2.70	100/ 2.65

We stock VERBATIM DISKS

Write for Dealer and OEM prices.

BASF 5" or 8"	10/ 2.00	20/ 1.95	100/ 1.85
NEW BASF Qualimetric Disks also in stock.			
Wabash 5"ss/sd	10/ 1.80	50/ 1.75	100/ 1.70
Wabash 5" ss/dd	10/ 2.00	50/ 1.95	100/ 1.90
Wabash 8" ss/sd	10/ 2.00	50/ 1.95	100/ 1.90

We stock MAXELL DISKS

Write for dealer and OEM prices.

Disk Storage Pages 10 for $5 Hub Rings 50 for $6
Disk Library Cases 8"—3.00 5"—2.25
Head Cleaning Kits 11

CASSETTES—AGFA PE-611 PREMIUM

High output, low noise, 5 screw housings.

C-10	10/ .61	50/ .58	100/ .50
C-30	10/ .85	50/ .82	100/ .70

SPECIALS

Zenith ZVM-121 Green Phosphor Monitor	100
BMC 12A 12" Green Monitor	80
VOTRAX Personal Speech System	280
VOTRAX Type-N-Talk	160
VOICE BOX Speech Synthesizer (Apple or Atari)	
Prowriter 8510 parallel	389
Okidata 92	520
Daisywriter 2000	1050
Many printers available (Star-Gemini, Brother, OKI, etc.)	
We Stock AMDEK Monitors	
Amdek DXY-100 Plotter	600
A P Products	15% OFF
Watanabe Intelligent Plotter 990 6-pen	1290
ISOBAR 4 Outlet Surge Suppressor/Noise Filter	49
We stock Electrohome Monitors	
dBASE II (8" format)	300
Panasonic TR-120M1P 12" Monitor (20 MHz)	149
Panasonic CT-160 Dual Mode Color Monitor	285

USI Video Monitors—Green or AMBER 20 MHz hi-res. Dealer and OEM inquiries invited

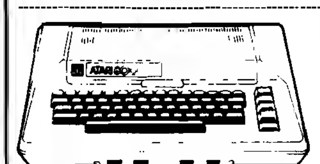
ALL BOOK and SOFTWARE PRICES DISCOUNTED

Synertek SYM-1 Microcomputer SALE 189
KTM-2/80 Synertek Video and Keyboard 349

ZENITH data systems

Z29 Terminal (new detached keyboard)	680
ZT-1 Intelligent Communications Terminal with Monitor	479
Z100 16-bit/8-bit System	CALL

We stock entire Zenith line.



ATARI SPECIALS

WE STOCK ENTIRE LINE—write for prices.

Atari 1200	749	**QIX**	34
Voice Box	100	**Anchor Modem—Atari**	79
Inside Atari DOS	18	**Atari Graphics** (Compute!)	11
Thorn EMI Software		**First Book of Atari**	11
EduFun Software		**APX Software**	

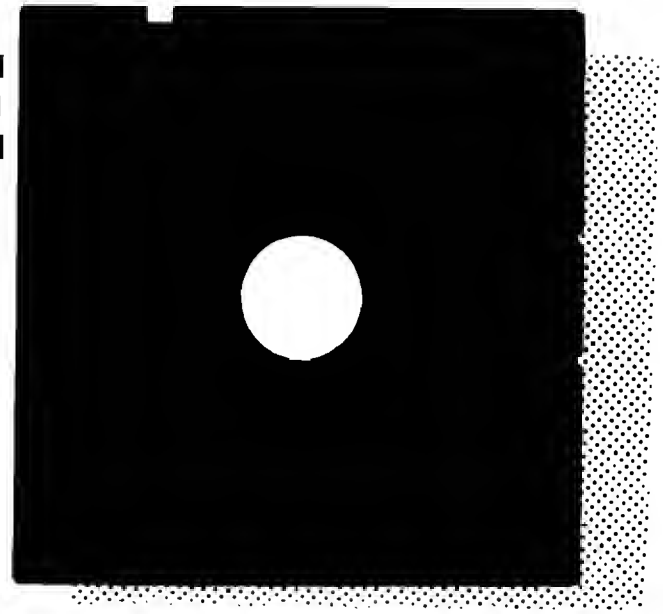
252 Bethlehem Pike
Colmar, PA 18915
215-822-7727

A B Computers

WRITE FOR CATALOG

Add $1.25 per order for shipping. We pay balance of UPS surface charges on all prepaid orders. Prices listed are on cash discount basis. Regular prices slightly higher. Prices subject to change.

GET MORE DATA ON YOUR CBM DISK



by Werner Kolbe

The Commodore operating systems write numerical data to disk as ASCII strings. This article shows you how to write up to four times more numerical data on the CBM disk using the computer's internal binary format instead of ASCII strings.

The data sheet for your floppy disk system tells you how many bytes you can store on a disk, but normally it does not tell you how effectively the system uses the space to store your data. In the case of the PET/CBM disks up to four times more data can be stored when a compressed format is used for storing numbers.

I use my Commodore PET in conjunction with a 4040 floppy disk system to store huge amounts of statistical data upon dogs' tooth failures. These data are mostly integers and only a few are large enough to require floating-point format. I noticed that these data, after they were written on the disk, needed more bytes than they had used in my computer. The reason is simple: the data files on disk or tape are represented in ASCII format, whereas in the computer they are stored more compactly. Storing the data in a compressed form is not less reliable than storing it in the ASCII code. In both cases you will get a wrong number (or read error) if the wrong byte is read from disk.

The operating system of the PET treats all external devices in the same manner, and so it sends data to the disk in the same format as it sends data to the printer, which is the ASCII format. Thus, an integer that can be represented by two bytes may need as many as eight bytes. You could save three-quarters of the space by writing the data in the same compressed format on the disk as it is stored in the computer.

The simplest and fastest way to store compressed data is to assign the value of the number that will be stored on disk to a special variable and then PEEK the storage area of that variable. The BASIC interpreter allocates storage area to the simple variables (not arrays) in the sequence of their occurrence behind the BASIC program, and so it is a good idea to use the special variable as the first variable of your program. This variable will then be located directly behind the BASIC program and can be PEEKed easily, using the contents of RAM locations 42, 43 that point to the beginning of the variable list.

The same principle is used to read the data back. Using GET, you read the appropriate number of bytes from the disk, then POKE the number into the storage area of the special variable. The latter will return the value of the number.

Listing 1, Compress Example 1, shows you how this method works if you want to store integers on the disk. The special variable is called XX%, but it will work with other names as well. The % identifier must be there to ensure that its content is represented by only two bytes. The variable YY points to the first of these two bytes, which you must send to the disk to write a number. To read it back into the computer POKE the ASCII values of the characters received from the disk into the storage area of the dummy variable XX%. The example writes two files on the disk to demonstrate the difference in the space needed for the same numbers. Then the compressed numbers are read back into the computer to complete the process.

Listing 2, Compress Example 2, shows how the compression can be done for floating-point variables. The only difference is that now the special variable must be of the floating-point type and that five bytes are used to represent a number. This time the space used on the disk is about three times less.

Finally, listing 3 shows how you can write both compressed integers and floating-point numbers within the same program. This time the variable XX is second in the list and you get the location of its storage area by adding 7 to the pointer YY%.

The example programs will run on all ROMS of the PET/CBM, VIC, and Commodore 64 computers. I do not know how other personal computers store numbers on disk, but if it is done in ASCII then a similar procedure could be used to increase the available space on disk.

You may contact Werner Kolbe at Vd Kamlaan 65, 2625 Kn Delft, Netherlands.

LISTING 1

```
100 REM EXAMPLE HOW TO COMPRESS INTEGERS
110 REM --------------------------------
120 REM
130 REM TWO SAMPLE FILES ARE WRITTEN.
140 REM ONE IN STANDARD ASCII FORMAT
150 REM AND ANOTHER IN COMPRESSED FORMAT.
160 REM THE COMPRESSED VERSION NEEDS ABOUT
170 REM FOUR TIMES LESS SPACE ON THE DISK
180 REM
190 REM XX% IS THE DUMMY VARIABLE
200 REM WHICH MUST BE THE FIRST VARIABLE
210 REM IN THE PROGRAM.
220 REM
230 REM YY POINTS TO THE PLACE WHERE
240 REM THE VALUE OF XX% IS STORED IN
250 REM INTERNAL COMPRESSED FORMAT.
260 REM
270 XX%=0
280 YY=PEEK(42)+256*PEEK(43)+2
290 REM
300 REM
310 OPEN1,8,3,"@1:STANDARD,S,W"
320 OPEN2,8,4,"@1:COMPRESSED,S,W"
330 REM
340 A=15E3
350 REM
360 REM WRITE 1000 NUMBERS
370 REM
380 FORI=ATOA+1E3
390 REM
400 REM 1. USING THE STANDARD METHOD
410 REM
420 PRINT#1,I;CHR$(13);
430 REM
440 REM 2. IN COMPRESSED FORMAT
450 REM
460 REM ASIGN IT TO THE DUMMY VARIABLE
470 XX%=I
480 REM
490 REM AND WRITE PETS INTERNAL REPRESENTATION
500 REM AS CHARACTERS ONTO THE DISK
510 REM
520 FORJ=0TO1:PRINT#2,CHR$(PEEK(YY+J));:NEXTJ
530 NEXT I
540 CLOSE1:CLOSE2
550 REM
560 REM
570 REM HERE I SHOW HOW TO READ BACK
580 REM THE COMPRESSED DATA.
590 REM
600 OPEN2,8,4,"1:COMPRESSED,S,R"
610 REM
620 REM WE GET A PAIR OF TWO CHARACTERS
630 REM AND POKE IT INTO THE STORAGE
640 REM AREA OF XX%.
650 REM
660 FORJ=0TO1
670 GET#2,A$:IFA$=""THENA$=CHR$(0)
680 POKE(YY+J),ASC(A$)
690 NEXTJ
700 REM
710 REM XX% NOW CONTAINS THE NUMBER
720 REM WHICH WE CAN USE.
730 REM
740 PRINTXX%:IFST=0THEN660
750 CLOSE2
READY.
```

COMPRESS
requires:
PET, CBM, VIC, or
Commodore 64 and a
Commodore disk drive

LISTING 2

```
100 REM EXAMPLE HOW TO COMPRESS FLOATING
110 REM POINT NUMBERS.
120 REM
130 REM
140 REM XX IS THE DUMMY VARIABLE
150 REM
160 REM AGAIN WE WRITE 1000 NUMBERS
170 REM IN STANDARD AND IN COMPRESSED
180 REM FORMAT ONTO THE DISK
190 REM
200 REM
210 XX=0:YY=PEEK(42)+256*PEEK(43)+2
220 OPEN1,8,3,"@1:STANDARD,S,W"
230 OPEN2,8,4,"@1:COMPRESSED,S,W"
240 FORI=1TO1000
250 A=RND(1)/100
260 PRINT#1,A;CHR$(13);
270 XX=A
280 FORJ=0TO4:PRINT#2,CHR$(PEEK(YY+J));:NEXTJ
290 NEXT I
300 CLOSE1:CLOSE2
310 REM
320 REM AND HOW TO READ IT BACK
330 REM
340 OPEN2,8,4,"1:COMPRESSED,S,R"
350 FORJ=0TO4
360 GET#2,A$:IFA$=""THENA$=CHR$(0)
370 POKE(YY+J),ASC(A$)
380 NEXTJ
390 K=XX:PRINTK:IFST=0THEN350
400 CLOSE2
READY.
```

LISTING 3

```
100 REM EXAMPLE HOW TO COMPRESS BOTH
110 REM INTEGERS AND FLOATING POINT
120 REM NUMBERS.
130 REM
140 REM XX% AND XX ARE DUMMY VARIABELS
150 REM
160 REM YY% AND YY ARE THE POINTERS TO
170 REM THEIR STORAGE AREA.
180 REM
190 XX%=0:XX=0
200 YY%=PEEK(42)+256*PEEK(43)+2
210 YY=YY%+7
220 REM XX IS THE SECOND VARIABLE
230 REM
240 OPEN2,8,4,"@1:MIXTURE,S,W"
250 FORI=1TO100
260 A=I+RND(1)
270 REM WRITE IN PAIRS I,A
280 REM FIRST I AS COMPRESSED INTEGER
290 REM
300 XX%=I:FORJ=0TO1:A$=CHR$(PEEK(YY%+J)):PRINT#2,A$;:NEXT
310 REM
320 REM AND THEN A
330 XX=A:FORJ=0TO4:PRINT#2,CHR$(PEEK(YY+J));:NEXTJ
340 NEXT I
350 CLOSE1:CLOSE2
360 REM
370 REM AND HOW TO READ IT BACK
380 REM
390 OPEN2,8,4,"1:MIXTURE,S,R"
391 REM
392 REM FIRST THE INTEGER
393 REM
400 FORJ=0TO1
410 GET#2,A$:IFA$=""THENA$=CHR$(0)
420 POKE(YY%+J),ASC(A$)
430 NEXTJ
440 I=XX%
441 REM
442 REM AND THEN THE FLOATING POINT
443 REM
450 FORJ=0TO4
460 GET#2,A$:IFA$=""THENA$=CHR$(0)
470 POKE(YY+J),ASC(A$)
480 NEXTJ
490 A=XX:PRINTI,A:IFST=0THEN400
500 CLOSE2
READY.
```

MICRO

Parameter Passing in Assembly Language

Part I

A discussion of several methods of passing data to and from assembly-language subroutines. 6502, 6809, 68000, and 16032 microprocessors are used for programming examples.

by Randall Hyde

Programming in assembly language has long been thought a black art accomplished only by skilled magicians and sorcerers. An assembly-language program requires many more statements to accomplish the same goal than does the equivalent Pascal or BASIC program. Because of the extra tedium involved, most people shy away from assembly language because there is too much to learn. In reality, anyone with enough patience can master assembly language easily. One of the major impediments to learning assembly language is learning to pass data to and from assembly-language subroutines. This article discusses various methods of passing data on the 6502, 6809, and the newer 6800 and 16032 microprocessors.

Passing Data in the Microprocessor Registers

The simplest method used to pass data to and from an assembly-language subroutine is *via* the processor registers. Any register capable of storing retrievable data is usable for passing data to and from an assembly-language subroutine. On the 6502 data can be passed in the accumulator, X register, Y register, and the processor status register. On the 6809 data can be passed in the A or B accumulator, the X or Y index register, or the U stack pointer register. On the 16032 data can be passed in any of the eight general purpose registers, the static base register, or the frame pointer. Of the four microprocessors described here, the 68000 provides the richest complement of processor registers. Eight general purpose and seven address registers are available to the programmer in addition to the 68000 stack pointer.

The 6502 A, X, and Y registers are eight bits wide, and so you are limited to transferring a single byte within one of the registers. Single-bit boolean/binary values can be passed in the carry, zero, minus, and overflow flags. Such variables can be tested easily using the 6502 branch instructions.

The 6809 A and B accumulators are both eight bits wide and can be used for passing 8-bit data to and from a 6809 subroutine. If you need to transfer 16 bits of data, the A and B accumulators can be concatenated to form a single 16-bit accumulator — the 6809 D accumulator. Sixteen-bit data also can be passed in the 6809 X, Y, and U registers. Single-bit data can be passed in the 6809 zero, carry, overflow, and negative flags. Thirty-two-bit data can be passed by passing the low order 16 bits in one register and the high order 16 bits in a second register.

The 16032 microprocessor from National Semiconductor is far more advanced than its 8-bit brothers, the 6502 and 6809. Almost all of the 16032 registers are 32 bits wide, allowing you to pass a considerable amount of data in a single register. Furthermore, there are eight general purpose registers, three special purpose registers, and the pro-

cessor status register available for use. Since a considerable amount of data can be passed using the 16032 registers, passing parameters in the registers is much more practical on the 16032 than on the 6502 or 6809.

When it comes to passing data in the processor registers, the 68000 wins hands down. With 15 32-bit undedicated registers and a 32-bit stack pointer, the 68000 provides more registers than any popular microprocessor currently on the market.

Deciding when data is to be passed in one of the processor registers is the only problem left to solve. Since registers are a scarce commodity, judicious use of the register resources is recommended. Suppose you need to pass three bytes of data to a 6502 subroutine. Since you have three registers available, it would seem natural to pass the parameters in the A, X, and Y registers. In most cases such a choice would be very poor because these registers may be required for calculations going on outside the subroutine being called. For example, assume you are passing three bytes of data to a 6502 subroutine in the A, X, and Y registers. Furthermore, suppose you wish to call this subroutine repeatedly (i.e., from within a loop). The 6502 X and Y index registers are perfect for controlling loops and variables external to some subroutine. Unfortunately the X and Y registers will not be available for such use if they must be used for passing parameters to a subroutine, especially since you cannot push the X or Y register onto the stack without disturbing the contents of the accumulator. For this reason the use of the 6502 X and Y registers for passing parameters should be avoided as much as possible.

The 6809's register set is much more capable of handling parameters passed in the register than is the 6502. First, the 6809 has more registers than the 6502; second, the registers are twice as wide as the corresponding 6502 registers. Nevertheless, you should avoid passing parameters in the X, Y, and U registers for the same reasons outlined in the preceding paragraph.

The NS16032, with its eight general purpose registers, removes many of the restrictions on passing parameters in registers. Still, the frame pointer and Static Base register shouldn't be used for passing data to and from subroutines. That leaves the eight general purpose 16032 registers. These registers (called R0...R7) are each 32 bits wide, and so the 16032 is capable of passing over ten times the data in the CPU registers that the 6502 can! There's only one problem with using the 16032's general purpose registers to pass parameters: they *are* general purpose. This means that any of the eight registers can be used as accumulators, pointers, counters, or index registers. On the 6502 and 6809 the X and Y registers are special purpose index, pointer, or counter registers. Therefore, as you are writing a program, you know what the intended purpose of these registers is. Caution must be exercised with the 16032 because any of the registers can be used as an index register.

The 68000's register set provides so many registers for the programmer to use that many of the comments pertaining to register usage in the previous paragraphs simply don't apply. The 68000 data registers are most advantageously utilized for passing pure data (since most of the arithmetic and logical operations work best with the data registers) and the 68000 address registers should be used for passing pointer data since these registers are best suited for address manipulation.

Once you understand the disadvantages of passing data in the processor registers (i.e., there are only so many of them and they may not always be available for parameter usage) the next step is to decide when and how data is to be passed in the processor registers. The most obvious candidate is a routine that requires a single byte to be passed to it or returns a single byte upon completion. A good example is the READ CHARACTER and WRITE CHARACTER routines found in most monitors and operating systems. The WRITE CHARACTER subroutine (PUTC) usually requires a single character written to the system console. The READ CHARACTER subroutine (GETC) usually returns a single character read from the system terminal.

On the 6502 the accumulator is the best choice when data needs to be transferred to or from a subroutine. On the 6809 the A or B accumulator can be used to transfer the data. On the 16032 any of the eight general purpose registers can be used, although R0 (for consistency's sake) should be used.

When additional byte parameters must be passed to a subroutine, additional registers must be consumed. On the 6502 the Y register should be used first (because the X index register is used for counting more frequently than the Y register), and the X register last. The 6809 has two 8-bit accumulators that can be used for passing 8-bit parameters. Once these are exhausted you should use the Y register, followed by the X register, followed by the U stack pointer. Remember, the X, Y, and U registers are all 16 bits wide so two bytes can be passed in these registers by packing and unpacking the data. Packing two bytes into one of these registers is accomplished easily by loading the A and B accumulators with 8-bit data and then transferring the 16-bit D accumulator to one of the index registers. Of course there may be a more direct way, such as loading the 16-bit register with an immediate value or loading from some 16-bit memory location.

When additonal byte parameters must be passed to a subroutine additional registers must be consumed. On the 6502 the Y register should be used first...the X register last.

The 68000 and 16032 provide so many general purpose registers that there is little chance of running out of them. On these processors you can simply use additional registers to pass the extra parameter bytes. Although the 68000 and 16032 sport a complete set of 32-bit registers, packing and unpacking the data to preserve space is inefficient; there are better methods for passing parameters.

When you need to pass 16-bit values

to or from a subroutine, the 6502's limitations quickly become apparent. The 6502's registers are all eight bits wide, so you must use two registers to pass a 16-bit value. Most contemporary programs pass the low order eight bits in the Y register and the high order eight bits in the accumulator. Sometimes it would be more convenient to pass the low order byte in the accumulator (because arithmetic and logical operations take place in the accumulator and the low order byte is always operated on first), but my experience has proven that consistency (even with other people's code) is more important than convenience. If you need to pass more than one 16-bit value on the 6502, you will have to resort to another technique for parameter passing as the 6502 has only three bytes of register storage.

Passing 16-bit data on the 6809 is much simpler than on the 6502. The 6809 features a full set of 16-bit data and index registers. To pass a single 16-bit datum, your best bet is to use the 6809 D accumulator (which is a concatenation of the A and B 8-bit accumulators). If you need to pass two 16-bit values, the D accumulator and Y index registers are your best bet. The Y register should be used in place of the X register because most 6809 programmers use the X register more often than the Y register (this is a carry-over from the 68000 days). If more than three words of data need to be transferred you should choose a method other than using registers to transfer the parameters.

The 68000 and 16032 provide quite a few 32-bit registers, so passing 16- or even 32-bit data to a subroutine is no problem. In general, however, no more than four pieces of data should be transferred in the processor registers because these processors support more advanced methods of parameter transfer.

If you are using the 6502 you should try to limit yourself to two bytes of data passed in the 6502 registers (be it two 8-bit values or a single 16-bit value). On the 6809 you should limit yourself to three 8-bit values or two 16-bit values (using A, B, and Y registers). On the 68000 or the 16032 you should use registers to pass parameters only if you are passing less than five parameters. If you must exceed these limtis, then you should use one of the other methods described in this article.

One of the major problems with passing parameters in a fixed location is that you must move the data into the fixed location before calling the subroutine.

Passing Parameters in a Fixed Location

Once your register list is exhausted, one of the easiest ways to pass data is by placing it in some fixed memory location before calling the appropriate subroutine. This method is quite popular with several monitor programs such as the Apple monitor and the Atari operating system (PEEKs and POKEs to BASIC hackers). By using this method of data transfer a virtually unlimited number of parameters can be passed. All you need to do is load up the appropriate memory locations and call the subroutine.

There are two principle disadvantages to this method of parameter transfer. First, loading data into a register and then storing it into some memory location requires more code and takes longer to execute than simply loading a CPU register with the parameter. Secondly, if you reserve a memory location for use by a subroutine (parameter or local variable) it cannot be used for anything else unless you are absolutely certain that no conflict will arise. Good programming practices dictate that you do not use such memory locations for anything else, otherwise the code will not be as maintainable as it could be.

One advantage to this method is that all three processors mentioned in this article support some sort of abbreviated addressing (zero page on the 6502, direct page on the 6809, absolute with a short displacement on the 68000 and 16032) that lets you access certain memory locations using a shortened instruction. Furthermore, accessing such absolute memory locations is usually faster than any other method of parameter transfer other than register usage.

Certain global values (such as the horizontal and vertical positions for the cursor in a video screen driver) are perfect for this type of parameter transfer since all code in the system can have immediate and easy access to the data. This method of parameter passing is also perfect for quick and dirty programs where the extra time and thought required to pass data in some other manner is too excessive to be justified (this is especially true on the 6502, which is rather poor with respect to parameter handling.

Passing the Address of a Parameter Block

One of the major problems with passing parameters in a fixed location is that you must move the data into the fixed location before calling the subroutine. In many applications (like calling a disk operating system OPEN command, or a similar routine) a considerable amount of data needs to be passed to the subroutine, but the parameter values remain static over the lifetime of a program. In these situations it is more efficient to pass the address of a parameter block instead of moving the parameters to some fixed location. One way to accomplish this is to pass the address of the parameter block in one or more of the CPU registers. This is how, for example, Apple's DOS expects the address of the IOB (input output block) to be passed to the RWTS subroutine. On the 6502, most people pass the low order byte of the address of the parameter block in the Y register and the high order byte in the accumulator. On the 6809 one of the index registers (probably the X index register) is the best choice for the job. On the 16032 any of the eight general purpose registers are fine for this type of parameter transfer.

Depending upon the circumstances, the above form of parameter addressing is quite useful if you have a set of stock parameter blocks you wish to pass to a subroutine. For instance, the RWTS example mentioned earlier could benefit from this type of parameter transfer. In your program you define several IOBs and switch between them simply by loading the address of the new IOB.

Example:

```
LDY #IOB1  ;GET IOB-LO
LDA /IOB1  ;GET IOB-HI
```

(Continued on next page)

```
        JSR  RWTS


        LDY  #IOB2
        LDA  /IOB2
        JSR  RWTS


        LDY  #IOB3
        LDA  /IOB3
        JSR  RWTS

        ETC


IOB1    BYT     IOB DATA HERE
IOB2    BYT     IOB DATA HERE
IOB3    BYT     IOB DATA HERE

        ETC
```

If the data in IOB1 differs greatly from the data in IOB2, and both of them differ from the data in IOB3, then this form of parameter transfer is much more efficient than moving the data into an IOB at a fixed memory location. Another advantage to passing this data is the fact that it does not tie up the use of some specific address in the CPU's memory space. All you need to pass is the address of the beginning of the parameter block and, therefore, it can be located anywhere in memory; in fact, it can be relocated at will. Note that any amount of data can be transferred using this technique; the only limitation is that you have enough memory to hold all the parameters. On the 6502 it is best to keep the size of the parameter block below 256 bytes due to the 8-bit limitation of the 6502 index registers. Rarely will a subroutine require more than 30-40 bytes of parameters, so this shouldn't be a problem.

Now that you can pass the address of the parameter to a subroutine, the only problem left is retrieving the actual parameters. While the 16032, 68000, and 6809 microprocessors let you access a data table indirectly through a register, the 6502 does not. To access the parameter data on the 6502 you must store the address into a pair of consecutive zero page memory locations and access the data table using the (ZP),Y or (ZP),X addressing modes. For example, assume that you have a parameter block of the form:

```
PBLOCK BYT PARM1
       BYT PARM2
       BYT PARM3
       ADR PARM4
```

and the subroutine has the calling sequence:

```
LDY #PBLOCK
LDA /PBLOCK
JSR  SUBROUTN
```

Inside "SUBROUTN" you would store the accumulator and Y registers into a pair of zero page variables (say PARMPTR); then you could access the parameters using the (ZP),Y addressing mode as follows:

```
LDY #0
LDA  (PARMPTR),Y  ;Fetches PARM1


LDY  #1
LDA  (PARMPTR),Y  ;Fetches PARM2


LDY  #2
LDA  (PARMPTR),Y  ;Fetches PARM3


LDY  #3
LDA  (PARMPTR),Y   GET PARM4-LO
INY
LDA  (PARMPTR),Y   GET PARM4-HI
```

On the 6809 accessing the parameter block is much easier. Assuming the address of the parameter block is passed in the X or Y register (it doesn't matter which one) the calling sequence would be:

```
LDX #PBLOCK      Get address of
                  parameter block
JSR SUBROUTN     Call subroutine
```

To access the data in the parameter block you would use the code:

```
LDA ,X            Access PARM1


LDA  1,X          Access PARM2


LDA  2,X          Access PARM3


LDD  3,X          Access PARM4
                   (16 bits)
```

The 68000 calling sequence and parameter access method is almost identical to that for the 6809. The calling sequence is:

```
MOVE.L  #PBLOCK,A0
JSR       SUBROUTN
```

Within the subroutine you would access data in the parameter block using statements of the form:

```
MOVE.B  0(A0),D0  ;Copy PARM1 into D0


MOVE.B  1(A0),D1  ;Copy PARM2 into D1


MOVE.B  2(A0),D2  ;Copy PARM3 into D2


MOVE.W 3(A0),D3  ;Copy PARM4 into D3
```

The 16032 calling sequence and parameter access mechanism is identical to that of the 68000. The calling sequence is:

```
MOVD     #PBLOCK,R7  ;Get address
                      ; of parameters
JSR       SUBROUTN
```

Within the subroutine you would access data in the parameter block using statements of the form:

```
MOVB  0[R7],R0  ;Copy PARM1 into R0


MOVB  1[R7],R0  ;Copy PARM2 into R0


MOVB  2[R7],R0  ;Copy PARM3 into R0


MOVW  3[R7],R0  ;Copy PARM4 into R0
```

To return data to the calling subroutine simply move the data into the parameter block. Since the calling program presumably knows the address of the parameter block, it can extract data easily from the parameter block after the subroutine returns.

Although the address of the parameter block was passed in the registers in these examples, this address is nothing more than a 16-bit (or 32-bit) parameter, which can be passed to the subroutine in any fashion possible. It could be passed in a register (as we have done here), in a dedicated memory location, or even from within some other parameter block! In fact, any method used for passing 16 or 32 bits of data to a subroutine can be used to pass the address of a parameter block to the subroutine.

(Continued on next page)

HDE inc.

BOX 120
ALLAMUCHY, N.J. 07820
201-362-6574

HUDSON DIGITAL ELECTRONICS INC.

THE TASK* MASTERS

HDE supports the *TIM, AIM, SYM and KIM (TASK) with a growing line of computer programs and peripheral components. All HDE component boards are state-of-the-art 4½" x 6½", with on board regulation of all required voltages, fully compatible with the KIM-4 bus.

OMNIDISK 65/8 and 65/5
Single and dual drive 8" and 5¼" disk systems. Complete, ready to plug in, bootstrap and run. Include HDE's proprietary operating system, FODS (File Oriented Disk System).

DM816-M8A
An 8K static RAM board tested for a minimum of 100 hours and warranted for a full 6 months.

DM816-UB1
A prototyping card with on-board 5V regulator and address selection. You add the application.

DM816-P8
A 4/8K EPROM card for 2708 or 2716 circuits. On board regulation of all required voltages. Supplied without EPROMS.

DM816-CC15
A 15 position motherboard mounted in a 19" RETMA standard card cage, with power supply. KIM, AIM and SYM versions.

DISK PROGRAM LIBRARY
Offers exchange of user contributed routines and programs for HDE Disk Systems. Contact Progressive Computer Software, Inc. for details.

HDE DISK BASIC
A full range disk BASIC for KIM based systems. Includes PRINT USING, IF . . . THEN . . . ELSE. Sequential and random file access and much more. $175.00

HDE ADVANCED INTERACTIVE DISASSEMBLER (AID)
Two pass disassembler assigns labels and constructs source files for any object program. Saves multiple files to disk. TIM, AIM, SYM, KIM versions. $95.00

HDE ASSEMBLER
Advanced, two pass assembler with standard mnemonics. KIM, TIM, SYM and KIM cassette versions. $75.00 ($80.00 cassette)

HDE TEXT OUTPUT PROCESSING SYSTEM (TOPS)
A comprehensive text processor with over 30 commands to format and output letters, documents, manuscripts. KIM, TIM and KIM cassette versions. $135.00 ($142.50 cassette)

HDE DYNAMIC DEBUGGING TOOL (DDT)
Built in assembler/disassembler with program controlled single step and dynamic breakpoint entry/deletion. TIM, AIM, SYM, KIM AND KIM cassette versions. $65.00 ($68.50 cassette)

HDE COMPREHENSIVE MEMORY TEST (CMT)
Eight separate diagnostic routines for both static and dynamic memory. TIM, AIM, SYM, KIM and KIM cassette versions. $65.00 ($68.50 cassette)

AVAILABLE DIRECT OR FROM THESE FINE DEALERS:

Progressive Computer Software
405 Corbin Road
York, PA 17403
(717) 845-4954

Johnson computers
Box 523
Medina, Ohio 44256
(216) 725-4560

Falk-Baker Associates
382 Franklin Avenue
Nutley, NJ 07110
(201) 661-2430

Perry Peripherals
P.O. Box 924
Miller Place, NY 11764
(516) 744-6462

Lux Associates
20 Sunland Drive
Chico, CA 95926
(916) 343-5033

Laboratory Microcomputer Consultants
P.O. Box 84
East Amherst, NY 14051
(716) 689-7344

Circle No. 27

Passing Parameters on the Stack

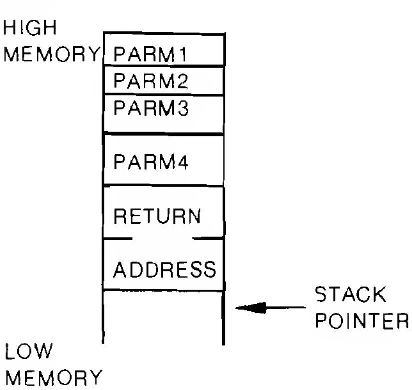
The favorite method for passing parameters used by many high-level languages such as Pascal is passing the parameters on a stack. The 6809, 68000, and 16032 are extremely well suited for passing data to and from procedures on a stack. The 6502 is, unfortunately, poorly suited for this application.

The 6502's problem with passing parameters on the stack is that the stack is of a very limited size (256 bytes). Furthermore, it's hard to get at data stored on the stack before a subroutine call. Nevertheless, there are times when pushing data onto the stack is the easiest way of passing data to a subroutine, especially if it is recursive. Although difficult, such parameter passing techniques are not impossible to perform on the 6502.

To pass data to a 6502 subroutine on the stack simply push the data onto the stack and call the routine as follows:

```
LDA  PARM1
PHA
LDA  PARM2
PHA
LDA  PARM3
PHA
LDA  PARM4
PHA
LDA  PARM4+1
PHA
JSR  SUBROUTN
```

There are two important points to keep in mind when passing data to a 6502 subroutine in this fashion. First, don't forget that upon entering the subroutine the subroutine return address is on the top of the stack. Second, the data appears on the stack in the reversed order (i.e., PARM4+1 is on the top, PARM4 is below that, PARM3 is below PARM4, etc.). Upon entering SUBROUTN the 6502 stack would look like this:



Note that the 6502 stack grows down and always points to the next available location on the stack. There are several ways to get the data off of the stack. One way is to POP it off (saving the return address in a temporary location and re-pushing it later) as follows:

```
SUBROUTN PLA
         STA  RTNADR
         PLA
         STA  RTNADR+1
         PLA
         STA  SPARM4+1
         PLA
         STA  SPARM4
         PLA
         STA  SPARM3
         PLA
         STA  SPARM2
         PLA
         STA  SPARM1
```

Push return address back onto the stack.

```
         LDA  RTNADR+1
         PHA
         LDA  RTNADR
         PHA



         RTS
```

This method of getting at parameters on the 6502 stack requires the use of some permanent memory locations. If you can't afford to use fixed memory locations to hold data, you can access it in place on the stack by copying the stack pointer into the X register and then using the index addressing mode:

```
TSX
LDA $107,X  ;Get PARM1


LDA $106,X  ;Get PARM2


LDA $105,X  ;Get PARM3


LDA $104,X  ;Get PARM4-LO


LDA $103,X  ;Get PARM4-HI
```

Remove parameters from stack and return.

```
PLA          ;Get LO return address
STA  $106,X  ;Save where rtn adrs
             ;must go
PLA          ;Get HI return address
STA  $107,X  ;Save at correct spot
PLA          ;POP parameters
             ;off stack
PLA
PLA
PLA
PLA
RTS
```

The routine shown here modifies the contents of the 6502 accumulator and X register. Most well-written subroutines will need to preserve the contents of the 6502 registers, which usually means they will need to be pushed onto the stack. If you push all three registers onto the stack, keep in mind that the offsets required to access PARM1, PARM2, PARM3, and PARM4 are increased by three. A better version of the former program that preserves all the 6502 registers is:

```
SUBRTN PHA                    ;Save Acc
       TXA
       PHA                    ;Save X
                              ;register
       TYA
       PHA                    ;Save Y
                              ;register
       TSX                    ;Get copy of
                              ;SP register

       LDA  $10A,X            ;Read
                              ;PARM1

       STA  $109,X            ;Store data
                              ;into
                              ;PARM2
       LDA  $108,X            ;Read
                              ;PARM3

       LDA  $107,X            ;Get LO
                              ;PARM4
                              ;value
       STA  PARM4SAV
       LDA  $106,X            ;Get HI
                              ;PARM4
                              ;value
       STA  PARM4SAV+1
```

Prepare for subroutine return

```
       LDA  $105,X            ;Get HI rtn
                              ;adrs byte
       STA  $10A,X            ;and
                              ;reposition
                              ;on the stack
       LDA  $104,X            ;Get LO rtn
```

```
                  ;adrs byte
STA  $109,X       ;and store
                  ;back onto
                  ;the stack
LDA  $103,X       ;Get Acc
                  ;value
STA  $108,X
LDA  $102,X       ;Get X
                  ;register
                  ;value
TAX
LDA  $101,X       ;Get Y
                  ;register
                  ;value
TAY
PLA               ;POP extra
PLA               ;junk off of
PLA
PLA
PLA               ;the stack
PLA
PLA
PLA               ;Get Acc
RTS               ;value
```

As you can probably tell, manipulating parameters on the 6502 stack is not a pleasant task.

The 6809 provides a much better method to access data on the stack than does the 6502. To begin, the 6809 supports *two* stacks: a hardware stack (where return addresses are kept) and a user stack (which is reserved for user-defined data). Both 6809 stack pointers can be used as index registers, so there's no need to waste one of the valuable index registers in order to access data on the stack. Finally, both stack pointers are 16 bits long so parameter data of any length (even when recursive subroutines are being used) can be accommodated.

While parameter data can be passed on either the hardware or the user stack, it is probably easiest to pass parameters on the user stack and use the hardware stack for storing return addresses and processor registers. Assuming that you have the same parameters defined for the 6502 example above, the calling sequence for a 6809 subroutine would be:

```
LDB   PARM1
LDA   PARM2
PSHU  A,B
LDA   PARM3
PSHU  A
LDX   PARM4
PSHU  X
JSR   SUBROUTN
```

After this code is executed the 6809 stack looks something like this:

```
HIGH
MEMORY   | PARM1 |
         | PARM2 |
         | PARM3 |
         | PARM4 |
LOW      | PARM4 |  <---- U
MEMORY
```

Here is the 6809 routine:

```
SUBRTN PSHS A,B,X,Y

       LDA    4,U      ;Get PARM1

       LDA    3,U      ;Get PARM2

       LDA    2,U      ;Get PARM3

       LDD    ,U       ;Get PARM4

       LEAU 5,U        ;Remove PARMs
                       ;from user
                       ;stack
       PULS A,B,X,Y    ;Restore 6809
                       ;registers
       RTS             ;All done
```

Notice that passing parameters on the 6809 stack is accomplished easily, especially when using the two stacks available to the user. Although it is a little more difficult to pass the parameters on the hardware stack (because the return address gets sandwiched in there) it is still much simpler to perform this task than it is to pass the data on the 6502 hardware stack. The 6809 code necessary to accomplish this task is:

```
SUBRTN  PSHS  A,B,X,Y ;Save 6809
                      ;registers

        LDA    12,S    ;Get PARM1

        LDA    11,S    ;Get PARM2

        LDA    10,S    ;Get PARM3

        LDD 8,S        ;Get PARM4

Get ready to leave

        LDD    6,S
        STD    11,S
        PULS   A,B,X,Y
        LEAS   5,S     ;Remove
                       ;parameter
                       ;storage
        RTS
```

The stack, after an invocation of this subroutine, looks like this:

```
| PARM1   |
| PARM2   |
| PARM3   |
| PARM4   |
| RETURN  |
| ADDRESS |
| Y(HIGH) |
| Y(LOW)  |
| X(HIGH) |
| X(LOW)  |
| B       |
| A       |  <---- Sp
```

The Motorola 68000 chip has eight address registers. A7 is the 68000 hardware stack pointer; the remaining seven registers can be treated like the user stack pointer on the 6809. The only difference between using the hardware stack pointer and any of the other address registers is that you must remember to respect the four-byte return address that is pushed onto the hardware stack during a subroutine call. The calling sequence for a 68000 subroutine is

```
MOVE.B PARM1,D0
MOVE.B D0,(SP)+     ;Note: SP is a
                    ;synonym for A7
MOVE.B PARM2,D0
MOVE.B D0,(SP)+
MOVE.B PARM3,D0
MOVE.B D0,(SP)+
MOVE.B PARM4,D0
MOVE.B D0,(SP)+
JSR    SUBRTN
```

Since the hardware stack is being used in this particular case, you must remember to index past the return address when fetching data from the stack. The code to accomplish this is

```
SUBRTN MOVEM A0-A6/  ;Save registers
             D0-D7,
             -(SP)

       MOVE.B 68(SP), ;Get PARM1
              D0
```

```
MOVE.B 67(SP),  ;Get PARM2
       D1

MOVE.B 66(SP),  ;Get PARM3
       D1

MOVE.W 64(SP),  ;Get PARM4
       D1

MOVE.L (SP),D0  ;Move return
                ;address down
MOVE.L D0,4(SP) ;four bytes
MOVEM  (SP)+,
       A0-A6/
       D0-D7
ADDA   #4,SP    ;Adjust stack
RTS             ;pointer
```

The National Semiconductor 16032 chip was designed with a stack architecture in mind. For this reason it handles parameters passed on the stack quite well. Assuming you have the same parameters as used in the previous three examples, the calling sequence for a 16032 subroutine would be:

```
MOVEB  PARM1,TOS
MOVB   PARM2,TOS
MOVB   PARM3,TOS
MOVW   PARM4,TOS
JSR    SUBRTN
```

You can see that it is much easier to push the data onto the stack with the 16032 than it was with the 6502, 6809, and 68000 microprocessors. The 16032 fully supports memory-to-memory data transfers, alleviating the problem of having to load the data into a register and then pushing it onto the stack.

Although the 16032 has only one hardware stack, the instruction set recognizes that the return address will be placed on top of the parameters and allows you to make adjustments for this. To access the parameters on the stack you would use the code

```
Save the NS16032 general purpose
registers and copy the stack pointer
into the 16032 frame pointer. This
also allocates zero bytes of local
storage for this particular subroutine.

SUBRTN ENTER [R0,R1,R2,R3,R4,
  R5,R6,R7],0

MOVB   6[FP],R0 ;Get PARM1

MOVB   5[FP],R0 ;Get PARM2

MOVB   4[FP],R0 ;Get PARM3

MOVW   2[FP],R0 ;Get PARM4

Now close up shop and quit

EXIT   [R0,R1,  ;Restore
        R2,R3,  ;registers
        R4,R5,  ;Return and pop
        R6,R7]  ;parameters
RTS    5
```

"Passing Parameters" will conclude next month.

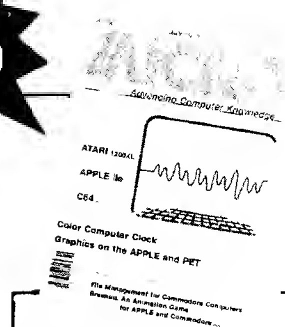
You may contact the author at Lazer MicroSystems, Inc., 1791 Capital Unit G, Corona, CA 91720.

MICRO

SAVE 20%!!!!
Subscribe to MICRO™

Use This Postage Paid Card to Order the Next 12 Issues of MICRO and SAVE $6.00 Off Newsstand Price!

MICRO™ SAVE 20%



☐ Yes! Enter my subscription to MICRO, and send me the next 12 issues for just **$24.00.** I save **$6.00 off** the newsstand price!

Name ____________________

Address ____________________

City ____________________

State ____________________ Zip ____________

I'm paying by ☐ Check ☐ MO ☐ VISA ☐ MasterCard

Card # ____________________ Exp.Date ____________

Signature ____________________

I OWN A:
☐ Commodore 64 ☐ VIC-20
☐ Apple ☐ PET
☐ Atari 400 ☐ Atari 800
☐ Other: ____________

For Faster Service Call:
1-800-345-8112
(In PA: 1-800-662-2444
VISA or MasterCard Only

A Feast Of Computing Ideas...

Order These Books From MICRO™

MICRO™ Books

NEW for VIC-20 Users!
Mastering Your VIC-20 With 8 BASIC Projects

A book that makes learning to program your VIC-20 fun! Contains 8 projects and programs. Games, utilities — even a VIC-20 version of "VisiCalc." All 8 programs on cassette to help you learn faster.

☐ Mastering the VIC-20 @ $23.95

NEW for OSI Users!
MICRO on the OSI

Includes Machine-Language enhancements and BASIC Aids, hardware modifications for enhanced/reversed video, programs for control code and upper/lower case entry. A valuable programming tool.

☐ Micro on the OSI @ $19.95

Best Sellers for APPLE Users!
MICRO on the APPLE

Programming aids, utilities, games, enhancements. Together Volumes 1, 2, and 3 contain over 100 programs on diskette. Fully documented and explained.

☐ 3-Volume Gift-Boxed @ $59.95
☐ Vol.1☐ Vol.2☐ Vol.3 $24.95 ea.

Please rush the MICRO Books I've checked above to:

Name ____________________

Address ____________________

City ____________________ State ________ Zip ____________

(Allow 6-8 weeks for delivery)

I'm paying by: ☐ Check ☐ MO ☐ VISA ☐ MC

Total Enclosed: $ ____________

(Add $2.00 s/h per book. MA res. add 5% tax)

Visa/MC # ____________________

Exp. Date: ____________________

MICRO™ Reader Service...

For More Information On The Advertisers In This Issue!

MICRO™ Reader Service Card

The numbers below correspond to the numbers imprinted on the advertisements in this issue of MICRO. This card valid for 90 days only.

Mail information to:

Name ____________________

Address ____________________

City ____________________ State ____________ Zip ____________

1	2	3	4	5	6	7	8	9	10	11	12	13
14	15	16	17	18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51	52
53	54	55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76	77	78
79	80	81	82	83	84	85	86	87	88	89	90	91
92	93	94	95	96	97	98	99	100	101	102	103	104

May 1983

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 60, CHELMSFORD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

SAVE 20%!!!!

Subscribe to MICRO

Use This Postage Paid Card to Orde[r] the Next 12 Issue[s] of MICRO and SAV[E] $6.00 Off Newsstand Price!

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 60, CHELMSFORD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

A Feast O[f] Computin[g] Ideas...

Order These Books From MICRO

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 60, CHELMSFORD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

MICRO Reader Service...

For More Information On The Advertisers In This Issue

Real Time Clock for Color Computer

by John Steiner

Use this subroutine to add a real-time clock and date function to your programs.

TIME$/SUB
requires:
TRS-80 Color Computer

After owning my TRS-80C for one year, I have few complaints and a lot of praise. The graphics are impressive for such a low-priced system, and the number of external hook-ups is more than adequate for the average user. Plus, you get the built-in RS-232 and joystick ports, which are extra cost on many other systems.

I finally purchased a disk drive and soon became accustomed to the invisible disk operating sytem. The DOS is an integral part of extended BASIC in the color computer. But I was jealous of my friends who owned the Model III with the built-in TIME$ and DATE$ commands.

One of the applications I have for my machine is a mailing list and file system for the local ham radio and computer clubs. It would be nice for a program to tell me the last time a file was created, or updated. I suffered with this shortcoming for many weeks until I saw the light.

The color computer has a built-in elapsed timer, which on power up counts from 0 to 65535. The timer is available only in Extended BASIC, and I have heard that it is interrupt driven, updating approximately every 1/60th of a second.

For the cost of a few extra reserved variables, I wrote a subroutine that can be inserted into any program. Listing 1 contains the routine, which begins at 60000.

You need to make only a couple modifications to any individual program to use the routine. After clearing memory and dimensioning strings, call the subroutine that begins at 60000, and immediately thereafter call the routine at 60100 as in the example.

```
10 CLEAR200:DIM N$(100)
20 GOSUB 60000: GOSUB 60100
```

The routine at 60000 enters the correct date and time into DATE$ and TIME$. Lines 60100 to 60160 update the correct time using the timer function.

Now, throughout the program, sprinkle GOSUB 60100 sparingly. Suitable places for the GOSUBs are at the beginning of menu screens, just before and after long sorts, and *always* just before printing TIME$ to the printer, screen, disk, or cassette.

The timer will keep reasonably correct time, unlike most software clocks, because it is interrupt driven.

When the routine is called, line 60100 determines whether or not one minute has expired. If it has not, no time is wasted, and the program reassigns or, in the case of initialization, assigns the current time to TIME$. If it has been a while since the last GOSUB, one minute is subtracted from the value of the timer and added to the variable MIN. If the hour must be updated, it is also done in line 60120. Another check is made to see if the timer is greater than 3550. This indicates that more than one minute has passed since the last update. If this is the case, the program loops back to 60110 and again updates the timer, hour, and minute variables.

You may find that the clock will gain or lose time. By changing the value of TIMER in lines 60100, 60110, and 60130, you can speed up or slow down the clock. You can make it as accurate as you want.

After the routine determines that the time is now correct, lines 60140 and 60150 format and assign the correct time to TIME$. Your program can get by without updating the time for 18 minutes, before the TIMER function in the color computer resets itself, so be sure that it is accessed more often than that. Again, it's a good idea to put the access routine in the menus, which are usually accessed often.

Now your computer has TIME$ and DATE$ capacity too!

***Listing 1:* TIME$/Subroutine**

```
50000 '*************************
50010 '*    TIME$/SUB           *
50020 '*  JOHN STEINER          *
50030 '* APRIL 11, 1982         *
50040 '*************************
50050 'TO CHANGE TIMER SPEED ADJUST
      LINES 60100,60110 AND 60130
60000 CLS:PRINT:PRINT"ENTER TODAYS DATE"
60010 LINEINPUT"MM/DD/YY: ";DATE$
60020 LINEINPUT"ENTER THE CORRECT HOUR: ";HOUR$
60030 IFVAL(HOUR$)<1ORVAL(HOUR$)>12THEN
      PRINT"INCORRECT TIME":GOTO60020
60040 HOUR=VAL(HOUR$)
60050 LINEINPUT"ENTER THE CORRECT MINUTE: ";MIN$
60060 IFVAL(MIN$)<0ORVAL(MIN$)>59THEN
      PRINT"INCORRECT MINUTES":GOTO60050
60070 MIN=VAL(MIN$)
60080 TIMER=0
60090 RETURN
60100 IFTIMER<3550THEN60140
60110 TIMER=TIMER-3550
60120 MIN=MIN+1:IFMIN>59THENMIN=MIN-60:
      HOUR=HOUR+1:IFHOUR>12THENHOUR=HOUR-12
60130 IFTIMER>3550THEN60110
60140 MIN$=STR$(MIN):IFLEN(MIN$)=2THEN
      MIN$="0"+RIGHT$(MIN$,1)ELSE
      MIN$=RIGHT$(MIN$,2)
60150 TIME$=STR$(HOUR)+":"+MIN$
60160 RETURN
```

John Steiner is an electronics instructor in the Fargo, ND school system. His hobbies include programming, amateur radio, and writing. He has written articles for several publications and is at present completing a book on Electronics. Also in the works is a book on BASIC. John's computer system includes a TPD-100 with disk system, and an Epson MX-80 printer.

MICRO

VIDEO TERMINAL BOARD 82-018

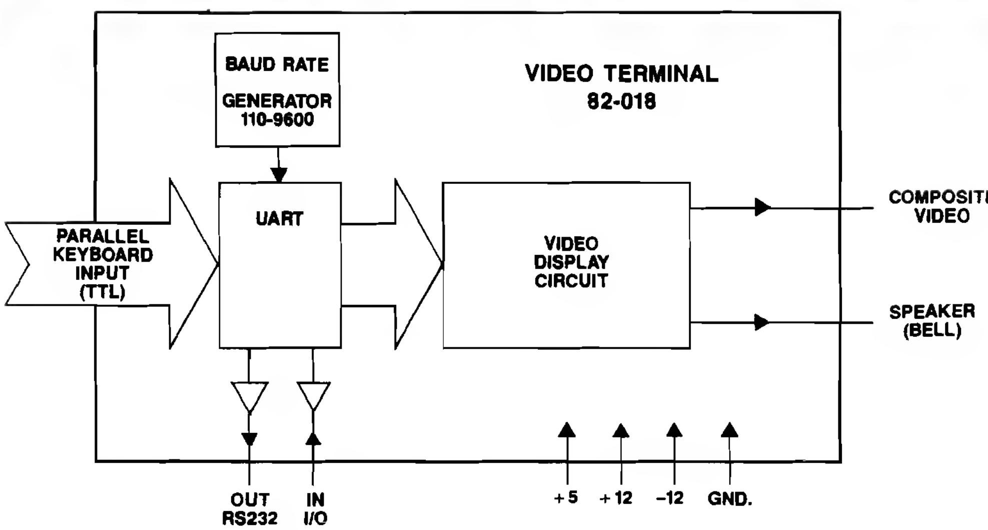
This is a complete stand alone Video Terminal board. All that is needed besides this board is a parallel ASCII keyboard, standard NTSC monitor, and a power supply. It displays 80 columns by 25 lines of UPPER and lower case characters. Data is transferred by RS232 at rates of 110 baud to 9600 baud — switch selectable. The UART is controlled (parity etc.) by a 5 pos. dip switch.

Complete source listing is included in the documentation. Both the character generator and the CRT program are in 2716 EPROMS to allow easy modification to your needs.

This board uses a 6502 Microprocessor and a 6545-1 CRT controller. The 6502 runs during the horz. and vert. blanking (45% of the time). The serial input port is interrupt driven. A 1500 character silo is used to store data until the 6502 can display it.

Features

- **6502 Microprocessor**
- **6545-1 CRT controller**
- **2716 EPROM char. gen.**
- **2716 EPROM program**
- **4K RAM (6116)**
- **2K EPROM 2716**
- **RS232 I/O for direct connection to computer or modem.**
- **80 columns x 25 line display**
- **Size 6.2″ x 7.2″**
- **Output for speaker (bell)**
- **Power +5 700Ma.**
  +12 50Ma.
  −12 50Ma.



This board is available assembled and tested, or bare board with the two EPROMS and crystal.

Assembled and tested	#82-018A	$199.95
Bare board with EPROMS and crystal	#82-018B	$ 89.95

Both versions come with complete documentation.

JOHN BELL ENGINEERING, INC.

ALL PRODUCTS ARE AVAILABLE FROM **JOHN BELL ENGINEERING, INC.** • 1014 CENTER ST., SAN CARLOS, CA 94070
ADD SALES TAX IN CALIFORNIA • ADD 5% SHIPPING & HANDLING 3% FOR ORDERS OVER $100
10% OUTSIDE U.S.A.
ADD $1.50 FOR C.O.D.

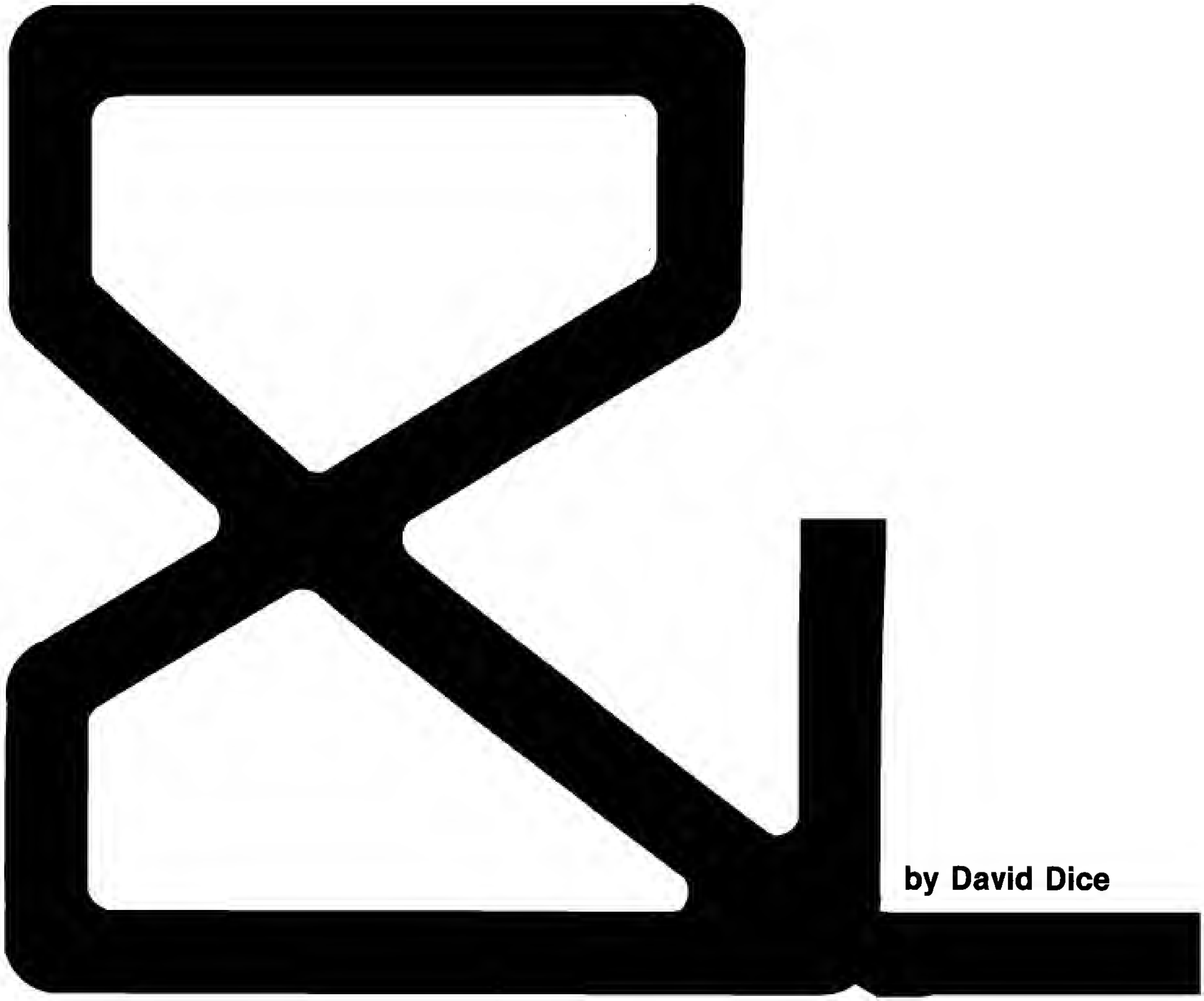
MasterCard

SEND $1.00 FOR CATALOG

(415) 592-8411
WILL CALL HOURS: 9am - 4pm

VISA

#249



by David Dice

A machine-language program to allow customized input routines.

AMPERGET
requires:
Apple II with Applesoft BASIC

Many functions are not available with Applesoft's INPUT routine. For instance, have you ever had to input strings containing commas or colons? Would you like to use the 'ESC' key as an exit parameter? Or how about using the backspace key to back up to a previous line of data? One possibility is using the GET routine, but this method has disadvantages. You can GET the character and then concantenate strings with a statement like GET X$: TE$ = TE$ + X$. However, you can no longer use the retype (— >) or backspace (< —) keys to copy or delete material from the string. Another problem is that it accelerates the need for garbage collection — that annoying, time-consuming process of freeing up memory that happens when you do a lot of string swapping in Applesoft. I tempered the problem by developing &GET.

Initially my approach to using the retype and backspace keys involved the MID$ command in a short Applesoft subroutine that found the correct part of the string. This worked at least as fast as I could type but there was some noticeable screen flicker. And because I was doing even more string manipulations, it compounded the problem of memory space!

When presented by a problem that BASIC doesn't solve efficiently you have two alternatives. Quit, or try another language! In this case I wanted the program to run quickly, so the logical approach was to write the input routine in machine language. Fortunately, the procedure could be greatly simplified by using a number of monitor and Applesoft routines and pointers. The result was &GET.

&GET takes about half a page of memory. It could be assembled in an area not affected by the Applesoft program that calls it, and page 3 of memory (starting at $300) is just such a convenient location. The &GET routine is accessed by a command using the syntax '&G,A$' where A$ can be any string variable. The & vector in byte $3F5 contains a JMP code to the beginning of the routine. The first few lines of the program check to ensure that the correct syntax was used and find the location of the string variable specified. Provided that the syntax is right the

routine begins to input the string. If there is an error, then you will hear the Apple's beep and be presented with the message 'SYNTAX ERROR'.

You put the characters you pick up in the input buffer, page 2 of memory. This is where the Apple's normal input routine, GETLIN, stores characters; since you're not using GETLIN there won't be any problems.

First the buffer is cleared by filling it with spaces. You are now ready to get the data. The monitor input routine, RDKEY, is used to input a character. If the character is one of 'ESC', ' ←', ' → ', or 'RETURN' then special action is taken:

1. If 'ESC', then \$1B (decimal 27), the ASCII code for ESCAPE, is placed in the variable and the routine exits.
2. If ' ← ', then the cursor is moved backwards and the length of the string is decremented. If an attempt

Amperget Listing

```
                        **********************************
                        *
                        * &GET
                        *
                        * COPYRIGHT (C) 1983
                        *
                        * MICRO INK
                        * 34 CHELMSFORD ST.
                        * CHELMSFORD, MA 01824
                        *
                        * WRITTEN BY D. DICE
                        *
                        **********************************
                        *
                        * APPLESOFT ROUTINES AND POINTERS
                        *
                        BASL     EPZ $28
                        CHKCOM   EQU $DEBE
                        CHKSTR   EQU $DD6C
                        CHRGET   EPZ $81
                        CROUT1   EQU $FD88
                        COUT     EQU $FDED
                        ERROR    EQU $D412
                        FRETOP   EPZ $6F
                        GETSPA   EQU $E452
                        IN       EQU $200
                        PTRGET   EQU $DFE3
                        RDKEY    EQU $FD0C
                        RNGBEL   EQU $FF3A
                        SYNTAX   EPZ $10
                        VARPT    EPZ $83
                        NULL     EQU $00
                        SPACE    EQU $20
                        FORWARD  EQU $95
                        BACKWARD EQU $88
                        RETURN   EQU $8D
                        ESCAPE   EQU $9B
                        AMPER    EQU $3F6
                        *
                                 ORG $300
                        *
                        * SET UP & JUMP
                        *
0300 A9 08                       LDA #BEGIN
0302 8D F6 03                    STA AMPER
0305 A9 03                       LDA /BEGIN
0307 8D F7 03                    STA AMPER+$1
030A 60                          RTS
                        *
                        * ENTRY POINT FROM AMPERSAND
                        *
0308 C9 47              BEGIN    CMP 'G          ;CHECK FOR 'G'
030D F0 05                       BEQ START
030F A2 10                       LDX #SYNTAX     ;SYNTAX ERROR
0311 4C 12 D4                    JMP ERROR
0314 20 81 00           START    JSR CHRGET
0317 20 8E DE                    JSR CHKCOM
031A 20 E3 DF                    JSR PTRGET
031D 20 6C DD                    JSR CHKSTR
0320 A2 00                       LDX #NULL       ; FILL THE BUFFER WITH SPACES
0322 A9 20              CLRBUF   LDA #SPACE
0324 9D 00 02                    STA IN,X
0327 CA                          DEX
0328 D0 F8                       BNE CLRBUF
032A 20 0C FD           GETCHAR  JSR RDKEY       ; READ THE KEYBOARD
032D C9 95                       CMP #FORWARD    ; CURSOR AHEAD?
032F F0 1A                       BEQ RDSCRN
0331 C9 88                       CMP #BACKWARD   ; BACKSPACE?
0333 F0 1B                       BEQ BACKWDS
0335 C9 8D                       CMP #RETURN     ; RETURN?
0337 F0 35                       BEQ RETURN1
0339 C9 98                       CMP #ESCAPE     ; ESCAPE?
033B F0 29                       BEQ ESCAPE1
033D 20 ED FD           PRINT    JSR COUT
0340 9D 00 02                    STA IN,X        ; STORE VALUE IN BUFFER
0343 E8                          INX             ; INCREMENT COUNT OF ENTRIES
0344 E0 F8                       CPX #$F8        ; IF GREATER THAN 248
0346 B0 50                       BCS BELL        ; THEN RING BELL
0348 4C 2A 03                    JMP GETCHAR     ; ELSE GET NEXT CHARACTER
0348 B1 28              RDSCRN   LDA (BASL),Y    ; READ THE SCREEN
034D 4C 3D 03                    JMP PRINT
0350 CA                 BACKWDS  DEX
0351 E0 FF                       CPX #$FF        ; IF < 0 THEN
```

is made to back up past the end of the line, then the variable contains the code for BACKSPACE, $8 and the routine exits.

3. If ' → ', then RDSCRN picks up the current character displayed on the monitor.
4. If 'RETURN', then the routine exits.

The Applesoft routine GETSPA is used to find space for the string variable, the high bit is stripped off, and then TSFR moves the input buffer to the variable specified in the &G,A$ statement. If the data is not one of these special characters, then it is added to the end of the input buffer. A check is made on the length of the input string. At 249 characters the bell begins to ring. At 255 characters the buffer has been filled, a backslash is printed, and the routine starts over.

The Applesoft program AMPERGET.DEMO (listing 2) shows how the routine can be used. The subroutine, GET THE DATA, uses the &GET. You may type in data, use the backspace and retype keys normally, and end a line by typing 'RETURN'. If you type a 'RETURN' without any entry, then the current value of the variable L1$ or L2$ is left unchanged. If you backspace past the beginning of INPUT LINE 2, then AMPERGET.TEST will take you back to INPUT LINE 1. Line 270 loops back to the beginning of the data input routine so that it is impossible to back up past the beginning of GET THE DATA. Typing 'ESC' at any time clears all the entries and repeats the GET THE DATA subroutine.

You lose the cursor editing functions ESC I,J,K,M (or A,B,C,D) by using this routine. This is a small price to pay for getting the ESC key back for an 'escape' function, and the ability to enter any character into an Applesoft string.

David Dice holds a Ph.D. in chemistry and a Bachelor's degree in education. He has taught chemistry, science, math, and computer science from the Junior High to the University levels. Mr. Dice is owner of Digipac Computer Consulting, a company specialzing in the production and distribution of quality educational and business software, and a fulltime high school teacher. You may contact him at Digipac, 907 River St. E., Prince Albert, Sask. 56V 0B3.

```
0353 F0 06              BEQ OUT          ; EXIT WITH BACKSPACE
0355 20 ED FD           JSR COUT         ; ELSE PRINT THE VALUE
0358 4C 2A 03           JMP GETCHAR
035B A2 00     OUT      LDX #NULL        ; EXIT WITH BACKSPACE
035D A9 08              LDA #BACKWARD-$80
035F 9D 00 02           STA IN,X
0362 E8                 INX
0363 4C 6E 03           JMP RETURN1
0366 A2 00     ESCAPE1  LDX #NULL        ; EXIT WITH ESCAPE CODE
0368 A9 1B              LDA #ESCAPE-$80
036A 9D 00 02           STA IN,X
036D E8                 INX
036E 8A        RETURN1  TXA              ; TRNSFR LEN TO ACCUM.
036F A0 00              LDY #NULL        ; AND STORE IT
0371 91 83              STA (VARPT),Y
0373 20 52 E4           JSR GETSPA       ; GET SPACE FOR STRING
0376 A0 01              LDY #$01
0378 A5 6F              LDA FRETOP
037A 91 83              STA (VARPT),Y    ; STORE ADDRESS OF STRING
037C C8                 INY
037D A5 70              LDA FRETOP+1
037F 91 83              STA (VARPT),Y
0381 A0 00              LDY #NULL
0383 B1 83              LDA (VARPT),Y
0385 AA                 TAX
0386 A8                 TAY
0387 CA        TSFR     DEX
0388 88                 DEY
0389 BD 00 02           LDA IN,X         ; GET CHARACTER
038C 29 7F              AND #$7F         ; CLEAR $80 BIT
038E 91 6F              STA (FRETOP),Y   ; STORE CHARACTER
0390 E0 00              CPX #NULL
0392 D0 F3              BNE TSFR
0394 20 8B FD           JSR CROUT1       ; EXIT AND CLEAR LINE
0397 60                 RTS
0398 E0 FF     BELL     CPX #$FF         ; IF STRING LEN EQUALS 255
039A F0 06              BEQ END          ; THEN DO AGAIN
039C 20 3A FF           JSR RNGBEL       ; ELSE  RING THE BELL
039F 4C 2A 03           JMP GETCHAR
03A2 A9 DC     END      LDA #$DC         ; PRINT SLASH AND START OVER
03A4 20 ED FD           JSR COUT
03A7 20 8B FD           JSR CROUT1
03AA 4C 22 03           JMP CLRBUF
03AD                    END
```

DICE AMPERGET.DEMO

```
10  REM ************************
20  REM *       AMPERGET.DEMO
30  REM *       BY DAVID DICE
40  REM *     COPYRIGHT (C) 1983
50  REM *        MICRO INK
60  REM ************************
70  ES$ =  CHR$ (27):CB$ =  CHR$ (
     8):D$ =  CHR$ (4)
80  PRINT D$;"BRUN AMPERGET.OBJ"
90  HOME
100  PRINT "'RETURN'=COPY 'ESC'=V
     OID '<-'=BACKSPACE"
110  VTAB 3: PRINT "INPUT LINE 1:
120  VTAB 13: PRINT "INPUT LINE 2:
130  GOSUB 170
140  IF ES THEN L1$ = "":L2$ = ""
     : GOTO 90
150  GOSUB 310
160  END
170 ES = 0
180  VTAB 3: HTAB 15: PRINT L1$
190  VTAB 3: HTAB 15: & G,A$
200  IF A$ = "" THEN A$ = L1$: VTAB
     3: HTAB 15: PRINT L1$
210  IF A$ = CB$ THEN 180
220  IF A$ = ES$ THEN ES = 1: RETURN
230 L1$ = A$
240  VTAB 13: HTAB 15: PRINT L2$
250  VTAB 13: HTAB 15: & G,A$
260  IF A$ = "" THEN A$ = L2$: VTAB
     13: HTAB 15: PRINT L2$
270  IF A$ = CB$ THEN 180
280  IF A$ = ES$ THEN ES = 1: RETURN
290 L2$ = A$
300  RETURN
310  HOME
320  PRINT "THE VALUES ENTERED WERE:
330  PRINT "LINE 1: ";L1$
340  PRINT : PRINT "LINE 2: ";L2$
350  RETURN
```

MICRO™

Modifying and Using MAE on the PET

by F. Arthur Cochrane

Modifications to the MAE assembler described here allow output to an ASCII printer and listing output to a disk file.

MAE is the Macro Assembler Text Editor for the PET written in fast machine-language code by Carl Moser of Eastern House Software. There are versions for 3.0 ROMs, 4.0 ROMs, and 8032 PETs. MAE has many powerful features, including macros and conditional assembly.

Patches to MAE for ASCII Printers

The following patches were provided by Carl Moser for ASCII printers:

1. To prevent sending a cursor down character before each line replace the following code with five NOPs:

```
74C4 A9 11        LDA #$11
74C6 20 53 72     JSR $7253
```

   It will look like this:

```
74C4 EA EA
74C6 EA EA EA
```

2. To eliminate line feeds make the following changes:

	From	To
6467	20	60
5553	0A 0A 0A	0D 0D 0D
556D	0A 0A	0D 0D
5571	0A	0D
5572	12	07
558D	0A	0D
5584	0A	0D
55A1	0A 0D 0A	0D 0D 0D
68F5	C9 0A	C9 0D

3. To output true ASCII instead of PET ASCII change 7210 20 B4 74 JSR $74B4 to 20 53 72 JSR $7253.
4. If you are using paper wider than 80 columns then the output can be widened slightly by changing the following to six NOPs (EA):

```
6027 C0 39     CPY #$39
6029 90 02     BCC $602D
602B A0 39     LDY #$39
```

5. If your printer is not device 4 then change 725A from 4 to your printer's device number.
6. To fix the STP (word processor) for no linefeeds to the printer make the following changes:

	From	To
0931	20 36 09	EA EA EA
0934	A9 0A	EA EA
0963	C9 0A	C9 0D
099F	C9 0A	C9 0D

7. To avoid losing an occasional character with any printer, enter these patches:

```
7256 85 2B 78
726D A5 2B 58
```

Send Listing Output to Disk

If you wish to send the listing to a disk file instead of a printer, then this patch will allow it. The following code tells the printer to listen.

```
7259 A9 04    LDA #$04
725B 85 D4    STA $D4
725D 20 D5 F0 JSR $F0D5 ($F0BA, Upgrade BASIC)
7260 20 48 F1 JSR $F148 ($F12D, Upgrade BASIC)
```

Change this code to the following to tell the disk to listen:

```
7259 EA EA
725B EA EA
725D EA EA EA
7260 20 F0 77    JSR $77F0
77F0 A9 08       LDA #8
77F2 85 D4       STA $D4
77F4 20 D5 F0    JSR $F0D5 ($F0BA, BASIC 2.0)
77F7 A9 68       LDA #$68
77F9 4C 43 F1    JMP $F143 ($F128, BASIC 2.0)
```

To use this disk output routine you must open the file in BASIC as OPEN8,8,8,"dr:name,s,w" then SYS to the MAE warm start ($5003). Now when the TO IEEE command is given the output goes to the open file. When you are through with the file give the BA command to return to BASIC and CLOSE8 to close the file.

Using the Relocating Loader

To start, all zero page locations that you want to be relocated should be assigned starting at location zero with the .DI pseudo-op. On the PET there are few locations in zero page to use. They are fixed and will be set with the .DE pseudo-op, but the .DI feature would be useful to develop software for another computer. The absolute (non-zero-page) locations used should be assigned with the .DI command starting at location $0400, but be sure to keep the same relative relationships among the labels that you desire in the final code. Now start assembly at location $2000. Note that MAE defaults the start of assembly at location $0400, not at location $2000, so you must explicitly use the .BA pseudo-op.

Now assemble the file with the AS command. After the two passes, give the OU command with a file name to put the relocatable file onto disk. Once the output command has done its pass the relocatable loader is ready to be used.

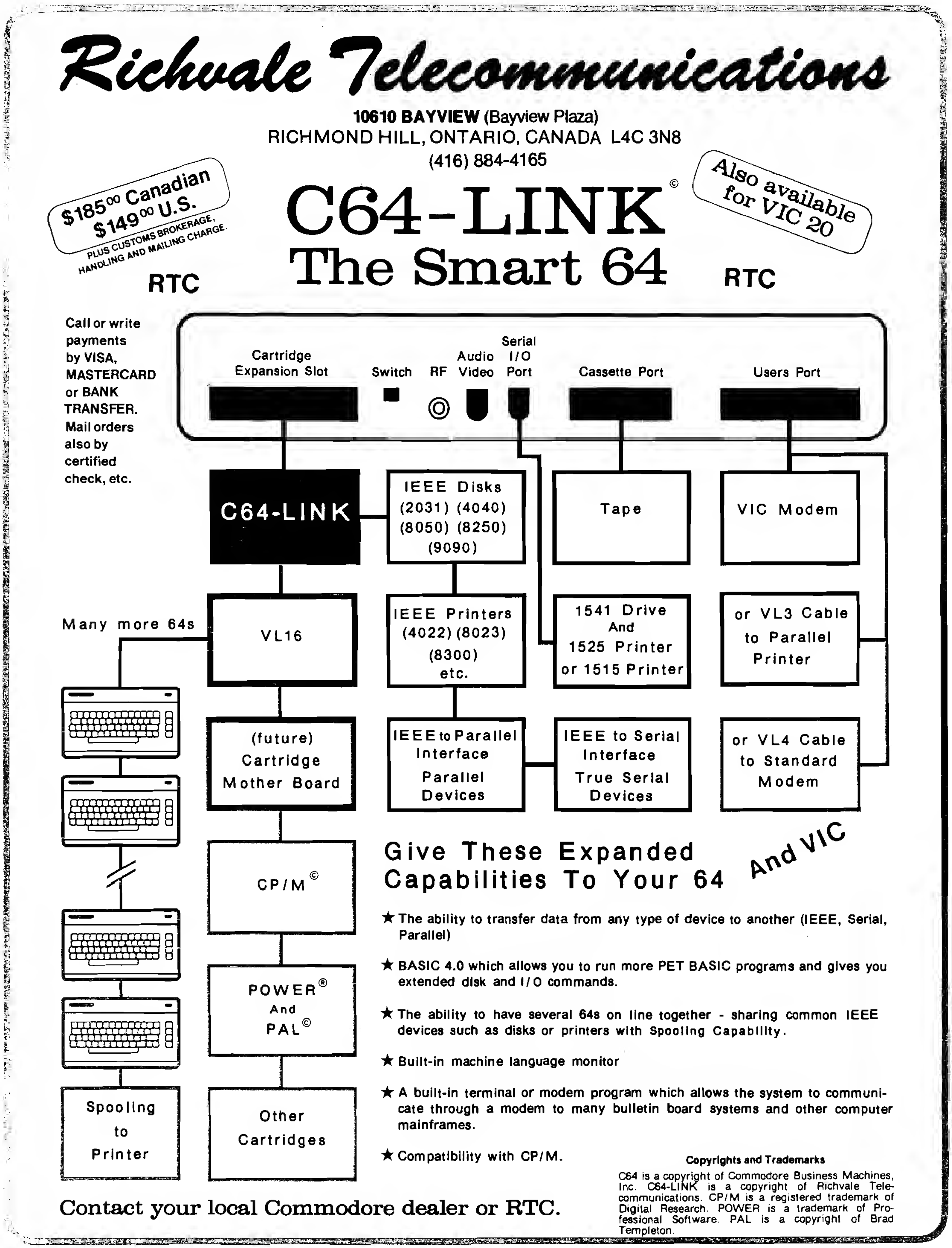
When the loader asks for the file name, give it the name of the file created with the OU command. For the zero-page offset and absolute offset enter numbers to add to the zero page and absolute locations (as defined with the .DI command). This will assemble them at their correct addresses. On the PET these will usually be zero. Next, for the program execution offset, give the location (in hex) where you want the program to run. For the program store offset, give the location (in hex) where you wish the program to be stored in memory. This allows programs to be stored in a different location than they will execute.

The relocating loader allows you to give someone only the relocatable file while retaining the source code. Here are examples of the output generated using the relocating loader.

```
FILE NAME? "0:REL.REL.04AUG"
Z-PG OFFSET? 0
ABS OFFSET? 0
PGM EXE OFFSET? 7800
PGM STORE OFFSET? 7800

LOAD MAP:
Z = 0000-0000
A = 0000-06BB
E = 7800-7800
S = 7800-7800
```

For information on the MAE user group send a SASE to ASM/TED Users Group (ATUG), c/o Brent Anderson, 200 S. Century, Rantoul, IL 61866, (217) 893-4577.

Mr. Cochrane has a Master of Science in electrical and computer engineering. Most of his programming on the PET is in assembler using the MAE assembler. You may contact him at 1402 Sand Bar Ferry Rd., Beech Island, SC 29841.


Richvale Telecommunications
10610 BAYVIEW (Bayview Plaza)
RICHMOND HILL, ONTARIO, CANADA L4C 3N8
(416) 884-4165
$185⁰⁰ Canadian
$149⁰⁰ U.S.
PLUS CUSTOMS BROKERAGE, HANDLING AND MAILING CHARGE.
C64-LINK ©
The Smart 64
Also available for VIC 20
RTC
RTC
Call or write payments by VISA, MASTERCARD or BANK TRANSFER. Mail orders also by certified check, etc.
Cartridge Expansion Slot
Switch
RF
Audio Video
Serial I/O Port
Cassette Port
Users Port
C64-LINK
IEEE Disks (2031) (4040) (8050) (8250) (9090)
Tape
VIC Modem
Many more 64s
VL16
IEEE Printers (4022) (8023) (8300) etc.
1541 Drive And 1525 Printer or 1515 Printer
or VL3 Cable to Parallel Printer
(future) Cartridge Mother Board
IEEE to Parallel Interface Parallel Devices
IEEE to Serial Interface True Serial Devices
or VL4 Cable to Standard Modem
CP/M ©
POWER ® And PAL ©
Other Cartridges
Spooling to Printer
Give These Expanded Capabilities To Your 64
And VIC
★ The ability to transfer data from any type of device to another (IEEE, Serial, Parallel)
★ BASIC 4.0 which allows you to run more PET BASIC programs and gives you extended disk and I/O commands.
★ The ability to have several 64s on line together - sharing common IEEE devices such as disks or printers with Spooling Capability.
★ Built-in machine language monitor
★ A built-in terminal or modem program which allows the system to communicate through a modem to many bulletin board systems and other computer mainframes.
★ Compatibility with CP/M.
Copyrights and Trademarks
C64 is a copyright of Commodore Business Machines, Inc. C64-LINK is a copyright of Richvale Telecommunications. CP/M is a registered trademark of Digital Research. POWER is a trademark of Professional Software. PAL is a copyright of Brad Templeton.
Contact your local Commodore dealer or RTC.

Circle No. 32

68000 INSTRUCTIONS

by Joe Hootman

Perhaps some of the most fascinating instructions implemented by the 68000 are those calssified as miscellaneous (see table 1).

Two trap instructions are available to the 68000 user. TRAP is a form of program control instruction; it loads the program counter with a particular vector and pushes the program counter and the status register onto the system stack. The TRAP instruction can select one of 16 trap vectors to be executed; any time the TRAP instruction is executed the machine will be vectored to the specified vector. The TRAPV instruction will perform in a similar manner to TRAP except that the V bit is checked; if it is set, the trap vector is fetched and executed. The TRAPV instruction is used extensively in two's complement arithmetic. When either of the TRAP instructions is executed, the processor goes into the exception processing state.

The Check Register Against Bounds (CHK) is used to compare the value in a register against a bound. The upper bound is expressed as a two's complement word-length integer located at the designated EA. The lower bound is zero. If the bound is exceeded, the CHK is selected and the processor goes into exception processing. This instruction allows the checking of array bounds by verifying that a data register contains a valid subscript. It is used extensively in higher-level languages such as Pascal.

Two special instructions are used to call subroutines. These instructions are the link (LINK) and Unlink (UNLK) instructions. The link statement is used to reserve space on the stack to be used by a called subroutine. The LINK instruction pushes the designated register onto the stack. The updated stack pointer is then loaded into the designated register. The specified 16-bit sign-extended displacement is then added to the stack pointer. This allows the subroutine to access the data area on the stack by indexing off the designated register.

UNLK is used to return the reserved data area. LINK and UNLK can be used to maintain a linked list of local data and parameters on the stack for nested subroutines. Generally the LINK state-

Figure 1

High Byte / Low Byte	
	Even Address
Operation word (opword)	word number 1
	Even Address + 2
Immediate data/operand (if any)	word number 2
	Even Address + 4
Source Effective Address extension (if any)	word number 3
	Even Address + 6
Destination Effective Address extension (if any)	word number 4
16 bits wide	

***Figure 2:* Opword Field Using a Single EA**

Opword Field Using a Single EA

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
X	X	X	X	X	X	X	X	X	X	Effective Address Mode	Effective Address Register

Figure 3

Opword Format Using Two EAs

15	14	13 12	11 10 9	8 7 6	5 4 3	2 1 0
X	X	Size	Effective Address Register	Effective Address Mode	Effective Address Mode	Effective Address Register

ment is used before the subroutine call and the UNLK is used just before the RTS at the end of the routine.

The NOP instruction just passes the control of the program to the next instruction.

Addressing Modes

Six general addressing mode categories are available to the 68000 user. Variations of the general categories extend the total number of addressing modes to 14. The 14 address modes combined with the 56 basic instructions give a potential of 784 separate instructions. However, some of the instructions do not make use of all the addressing modes. The addressing modes that are implemented in the 68000 are listed below.

1. Direct Addressing Using Register
   a. Data Register Direct
   b. Address Register Direct
2. Direct Memory Addressing
   a. Absolute Short
   b. Absolute Long
3. Indirect Memory Addressing
   a. Register Indirect
   b. Post-increment Register Indirect
   c. Pre-decrement Register Indirect
   d. Register Indirect with Displacement
   e. Register Indirect with Index and Displacement
4. Implied Register Addressing
5. Program Counter Relative Addressing
   a. PC-relative with Displacement
   b. PC-relative with Index and Displacement
6. Immediate Addressing
   a. Immediate
   b. Quick Immediate

The 68000 expects to find instructions in a prescribed sequence with the first 16-bit word being the operation word (opword). Every instruction must have an operation word; this word tells the processor what addressing mode is being used and how many extension words are associated with the instruction. Not every instruction needs to make use of the extension words. For example, the instructions that make use of implied addressing do not make use of the extension word.

The general format for the total instruction is given in figure 1.

A major difference between the 68000 and existing 8-bit processors is the way the 68000 accesses memory. The 68000 generally expects to fetch instructions from the memory on even addresses. If the address bus is forced to an odd address the processor will go

***Table 1:* Miscellaneous Instructions**

Mnemonic	Data Size/CCR	Name	Comments
NOP	— CCR X N Z V C - - - - -	No Operation	No operation occurs and execution continues with the next instruction. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 0 0 1
TRAP	— CCR X N Z V C - - - - -	Trap	This instruction forces the processor to one of 16 trap vectors. The processor will be put into the exception processing state. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 0 0 | Vector The vector field specifies which trap vector contains the new program counter to be loaded.
TRAPV	— CCR X N Z V C - - - - -	Trap on Overflow	This instruction initiates exception processing if the overflow condition is true. The new program counter is loaded from the TRAPV exception vector. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 0
CHK	16 CCR X N Z V C - • U U U	Check Register Against end Bounds	This instruction compares a specified data register to an upper bound. The upper bound, a 16-bit two's complement integer, is exceeded then the processor picks up the CHK exception vector and enters exception processing. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 | Register | 1 1 0 | Effective Address Mode | Register The field specifies the data register whose contents are to be checked. The effective address specifies the upper bound of the word to be checked against Dn. The following effective address modes cannot be used. 2, 13, 14.*
LINK	— CCR X N Z V C - - - - -	Link and Allocate	The following steps are executed: 1. The contents of the specified address register are pushed onto the stack. 2. The address register is then loaded with the updated SP. 3. A 16-bit sign-extended offset is added to the SP. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 0 1 0 | Register Displacement The register field specifies the address register through which the link is to be constructed; the displacement field specifies the two's complement integer that is to be added to the stack pointer.
UNLK	— CCR X N Z V C - - - - -	Unlink	The stack pointer is loaded with the specified address register and the address register is loaded with the long word data from the top of the stack. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 | Register The register field specifies the address through which the unlinking is to be done.

* See next month's table.

into exception processing. Exception processing will also occur if a word or long word operand fetch occurs on an odd address.

Most instructions make use of a single effective address (EA) in the operation word (opword) field. The format for the opword field is given in figure 2.

The Xs in the high order bits of the opword are encoded to indicate the type of operations to be performed. The EA defines the addressing mode, and if a single EA is used, the EA may be either a source or destination. The source is where the microprocessor will seek the data to be operated on, and the destination is where the data is stored after it has been operated on by the instruction.

The EA is made up of two parts — the mode and the register. The register number may be either a data register or an address register, depending on what is specified in the mode. The mode then specifies the type of addressing and register type to be used to implement the addressing mode. The coding of the opword for the various effective address (EA) modes can be coded by using the table provided next month. The single EA format is used for most of the opwords except for a few MOVE instructions. The move instructions that do not use the above format make use of two effective addresses in the opword. The format for these double EA is given in figure 3.

In addition to the effective address information in the opword, the size of the data or operand is specified if variable length data is permissible. The code for the "size" field is 00-byte (8 bits), 01-word (16 bits), 10-longword (32 bits).

To illustrate the addressing modes each mode will be discussed with an example. The MOVE instruction will be used for the examples as it is one of the most common instructions and uses all the addressing modes. The extension of the addressing modes to other instructions is relatively straightforward.

Editor's note: The second part of the discussion on Addressing Modes (including the table), will appear next month in MICRO.

Joe Hootman can be contacted at the University of North Dakota, Department of Electrical Engineering, University Station, Grand Forks, North Dakota 58202.

MICRO

It's All Relative Part 5

by Jim Strasma

This part of the series explains how to format and print data from Comodore relative disk files.

Before I begin this month, I have one correction for those without BASIC 4. In parts 2 and 3 of this series, (MICRO 56 and 57), I showed substitutes for BASIC 4 commands using the disk command channel: ie., OPEN 15,8,15:PRINT#15,''[command]'': CLOSE 15. I suggested it would be better to open file 15 to the disk once in setup, and leave it open. Arthur Cochrane reminds me that it's not only better — if you don't leave it open, CLOSE 15 closes all other disk files. BASIC 2 users should open file 15 at the start of the setup module and keep it open until ending the program in the menu module. Sorry for any inconvenience.

The obvious task neglected thus far is the one most people associate with a mail list — printing mailing labels. Although you now can create, find, display, and edit names and addresses with ease, you must be able to transfer those results to paper. For that reason a printer is one of the most useful devices to use with a computer, more important even than the disk drive. Let's put it to use.

Preliminaries

There are two kinds of printers — Commodore's and everyone else's. They speak ''PETSCII'' and ASCII,'' respectively. As these names imply, their character sets are similar but not identical. Some interfaces automatically convert PETSCII to ASCII. Of these, some do so whether you like it or not, and others allow the conversion to be switched off. Try to get the switchable kind. Many programs do PETSCII to ASCII conversion internally and can do fancier things with ASCII printers if the interface doesn't undo their work.

Bennett's Mail List does PETSCII to ASCII conversions on request. This is set up in the first module of the program by the variable PZ$. If it contains ''a'', conversion is done. If it contains ''p'', no conversion is done. If it contains ''n'', printing is to the screen only.

Although PZ$ is selected in the first module, most of its work is done in the ''4040 print'' module (my present example). If the printer uses ASCII, line 1110 swaps the preset mneumonic fields from the setup module to keep printouts in the proper case. Also, if the output is to the screen instead of a printer, lines 1120 and 1130 set up proper vertical spacing, using variable C$, which contains only a carriage return:

```
1110 IF PZ$=Z"A" THEN WK$=C1$:
     C1$=C3$:C3$=WK$:WK$=C2$:
     C2$=C4$:C4$=WK$
1120 CR$=" "
```

```
1130 IF PZ$ = "N" THEN CLOSE 4:
     OPEN 4,DV:CR$ = C$
```

Notice that line 1130 first CLOSEs then reOPENs file number 4. This is a good idea any time you are unsure whether or not the file is already open from use elsewhere in a program. When there is no printer, the program presets variable DV to 3, the device number of the screen. Thus, line 1130 essentially tells PET to use its own display screen as a printer. Then the user sees the first of several questions in a short menu:

```
1240 PRINT "PRINT THE MAILING LIST
     BY:"
1250 PRINT "1 LAST NAME & INITIALS
1260 PRINT "2 POSTAL CODES
1270 PRINT "3 CODES
1280 PRINT "4 ALTERNATE KEY
1290 PRINT
1300 PRINT " (USE -1 TO -4
     MEMBERS ONLY.)"
1310 MO = 0
1320 GOSUB 3470:REM GET
1130 IF WK$ = " - " THEN MO = 1:
     GOSUB 3470:REM GET
```

These choices are not exhaustive; with small changes, other fields could be read instead of the postal code. However, the choices are effective as they are and include not only the usual alphabetical output by key (1), numeric by record number (4), and all-important zip code order, but also a powerful code option and a further option that preselects by code regardless of which other option is chosen — the "-" option. The latter was added when I needed to send out a financial drive letter using a mail list consisting of both church members and occasional visitors. Not wanting to offend visitors, I preset the program to eliminate all but member records from the printout. To keep the method general-purpose, it is set up in the start-up module, where the test column within the code field and its allowable "member" contents are selected. For my use, column 1 must contain either an "M" or a "C" to be considered a member.

One other preliminary chore is the actual reading of the key file. If this is a rerun of the module, the key file may already be in memory. If so, why read it again? Variable GD (Got Data) keeps track of this:

```
1370 IF SB = GD THEN DOPEN#1,
     (F$),D(DD):GOTO 1420
1380 GOSUB 3220:REM READ KEY
     FILE
1390 GD = 0
1400 IF SB < > 1 THEN GD = SB:REM
     KEY READ
```

How the key file is read depends on the type selected. If the key is the primary or alternate key, its index file is read just as in the update module (MICRO 57:34). Otherwise, the selected field of data is read from each record into a new temporary index array that is retained in memory during the printout. This usually consumes more memory than reading an index file, so a calculation is made in lines 3790-3810 to limit the protion of the key that is stored in memory when it is necessary to conserve space:

```
3780 S2 = LE%(KF)
3790 WK = INT((FRE(0)-999)/NV)-2
3800 IF WK < 1 THEN WK = 1
3810 IF S2 > WK THEN S2 = WK
```

(C64 owners won't need to change line 3790. The program uses enough memory to keep it from reporting negative bytes free.)

Then line 3860 makes sure each key entry is the same length, using BL$ (a string of blanks) as a spacer:

```
3860 : K$ = MID$(D1$(KF) + BL$,S1,S2)
```

IT'S ALL RELATIVE
requires:
Any Commodore disk drive with DOS 2 or higher

Now it is necessary to insert the key entry in the key array being built in memory. The sort routine below is a variation of the binary search technique I used to locate a record (MICRO 57:33).

```
3870 : REM CALC POS
3880 : I = 1
3890 : J = V
3900 : IF I > J THEN 3960
3910 : K = INT((I + J + 1)/2)
3920 : IF K$ = KY$(K) THEN 3960
3930 : IF K$ < KY$(K) THEN J = K - 1:
       GOTO 3900
3940 : I = K + 1
3950 : GOTO 3900
3960 : P = K
3970 : V = V + 1
3980 : REM INSERT KEY INTO ARRAY
3990 : P = P - 2
4000 : IF P < 1 THEN P = 1
4010 : IF V = 1 THEN KY$(1) = K$
       :K%(1) = RR:GOTO 4130
4020 : FOR K = P TO V
4030 : : IF K$ < KY$(K) THEN
       P = K:K = V
4040 : NEXT
4050 : IF K$ < KY$(P) THEN 4090
4060 : KY$(V) = K$
4070 : K%(V) = RR
4080 : GOT 4130
4090 : E = V
4100 : SYS DL,0,P,E,KY$(0),K%(0),ZZ
4110 : KY$(P) = K$
4120 : K%(P) = RR
4130 : IF V = NV THEN RR = NR
4140 NEXT
4150 RETURN
```

Line 4100 keeps the time delay for this sorting process within reason by moving all necessary entries up one space in the array to leave room for the current key. As shown before, this could be done in BASIC, but far more slowly (MICRO 55:34).

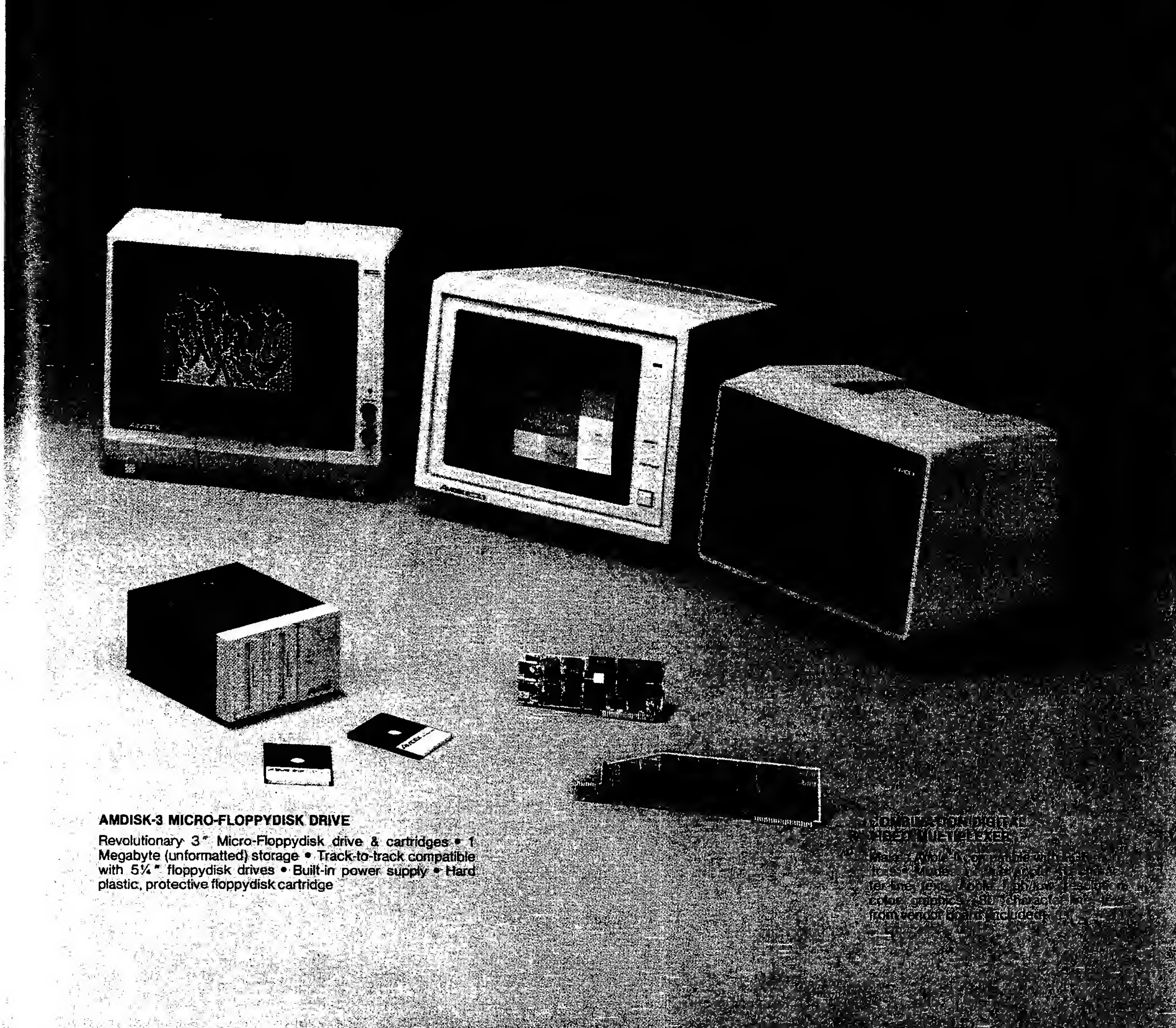
A second menu selection is offered after the key field has been read in. In this second menu, the possible printouts are listed:

```
1480 PRINT "0 = END
1490 PRINT "SELECT THE LISTING
     TYPE
1500 PRINT "1 PRINT MAILING
     LABELS
1510 PRINT "2 PRINT MAIL/PHONE
     LABELS
1520 PRINT "3 PRINT COMPLETE
     REPORT
1530 PRINT "4 PRINT TELEPHONE
     LIST
1540 PRINT "5 PRINT VISITING LIST
1550 PRINT "6 DUMP OUTPUT TO
     DISK
1560 PRINT "7 COUNT ONLY
1570 PRINT "(USE -1 TO -7 FOR
     LOCALS ONLY)
1580 GOSUB 33470:REM GET
1590 IF WK$ = "0" THEN 2930
1600 LO = 0
1610 IF WK$ = " - " THEN LO = 1:
     GOSUB 3470:REM LOCALS ONLY
```

As above, these options are not the only ones possible, but include all those needed in my two years of use. Most are variations of fields included in the printout. The disk option, however, is flexible. It allows any number of fields in any order to be sent to the disk as a sequential file. This allows the mail list to be used effectively with most popular word processors for form letters. It also eases the chore of restructuring data when sending it to another data-manager program.

The count-only option was added when I found myself wondering how selective to be in bulk mailings. In the U.S., non-profit bulk mailings must go to 200 persons to qualify for cut-rate postage. This option allowed me to try different selection criteria to see how many would qualify before wasting any mail labels *(Continued on page 67)*

innovative computing!



Amdek Corp. is dedicated to marketing quality computer peripheral equipment to enhance the use of popular personal computers. Our research & development staff keeps abreast of progress in computer techology and equipment and strives to offer you state-of-the-art advances in peripheral equipment.

Amdek products are distributed nationwide and in Canada through major distributors. And, we have factory-trained manufacturer's representatives ready to serve you in every major marketing area. Amdek offices are located in Chicago, Los Angeles & Dallas.

Just circle the reader service number, or contact us to receive complete technical specifications on these Amdek products.

AMDEK CORP.

2201 Lively Blvd. • Elk Grove Village, IL 60007
(312) 364-1180 TLX: 25-4786


EVERYONE NEEDS A . . .
BATTERY 6 MINUTES
V•AC LINE
BATTERY 2 MINUTES
GUARDIAN ANGEL™
RH ELECTRONICS, INC.
DVANCED DESIGN
EW RELIABILITY
G RDIAN PROTECTS
ELECTRONIC SAFETY
ER FOR DATA
$595
MONITOR —
MICRO COMPUTER
DISC DRIVE
SAVE YOUR DATA
• UL listed • FCC approved
• Green/red LED power status indicator • Green — normal AC line power • Slow blinking red — at least 6 minutes of remaining standby power • Fast blinking red — approximately 2 minutes of remaining battery power • Solid state technology unexcelled by any UPS power unit in its class.
RH ELECTRONICS, INC.
COPYRIGHT © 1981 - PATENTS PENDING
566 IRELAN, BUELLTON, CA 93427
(805) 688-2047
SEE YOUR RH ELECTRONICS PRODUCTS DEALER
FOR YOUR APPLE II*:
SUPER FAN II™ .... $ 74.95
SUPER FAN II™/ZENER RAY™ $109.00
SUPER RAM II™ $125.00
RH 12 VOLT TRANSVERTER $149.00
FOR MICRO COMPUTERS:
GUARDIAN ANGEL™ $595.00
Circle No. 40

Each option requires a bit of added setup at this point. Variable PT$ is set to the type of printout wanted. The code used lets "s" stand for a summary, "l" for label, "o" for output to disk, and so on. Notice that additional variables are set for some options. For example, ZT = 1 when you want a mail label plus phone number. This eliminates needing an extra routine to format it.

```
1360 IF SB = 3 THEN GOSUB 3540:
     REM SELECT CODE
1660 IF WK$ = "2" THEN PT$ = "L":
     ZT = 1
1670 IF WK$ = "4" THEN PT$ = "P"
1680 IF WK$ = "5" THEN PT$ = "V"
1690 IF WK$ = "6" THEN PT$ = "O":
     NU = 1:GOTO 1800
1700 IF WK$ = "7" THEN PT$ = "N":
     NU = 1:GOTO 1800
1710 IF PZ$ = "N" THEN NU = 1:FL = 1:
     GOTO 1800
```

The following section selects 1-up, 2-up, or 3-up output to match various paper sizes. One-up is preset in variable NU by options that require it.

Another common problem when printing mail labels is the printout that goes haywire halfway through. Bennett's Mail List offers some helps for this as you will see, provided you notice in time. If you don't notice until everything is done, you can re-run the print option and start the output from any point within the data. The program will ask "Start Printing from Beginning (y/n)." If you want to start somewhere else answer "n" and give the record number you want to use as a starting point.

If you have used an Addressograph™ to print mail labels, you may have noticed metal flags stuck to the edge of its printing plates. These allow the user to select certain labels from a group. Bennett's Mail List also allows this. It has 20 flags, each a character in the code field of the relative data file. These are more powerful than their Addressograph counterparts in that Bennett's flags may contain any alphabetic character. With careful selection of characters, one position may serve to order a large organization.

Because the code option is so important, it receives special handling in the print module setup. Regardless of other options already selected, the user also may select by code. Within the selected code position(s), 10 matches may be specified. There is also a "wild card" match-all option, which accepts anything other than a blank as a match. The wild-card character, normally "z", is set by the contents of MA$ in the startup module.

```
2080 PRINT "TO MATCH ALL NON-
     BLANKS, SELECT "MA$
2090 PRINT "OTHERWISE, LIST 1-10
     DESIRED MATCHES
2100 PRINT "NOW. THEN HIT [RETURN]
     ALONE
2110 FOR I = 1 TO 10
2120 : PRINT LEFT$(PO$,KL + I)
     "FIELD: ";
2130 : SYS IN,27,S2,L$
2140 : IF L$ = " " THEN CF = I - 1:
     I = 10
2150 : IF L$ = "     " THEN I = I - 1:
     IF I    0 THEN I = I - 1
2160 : CF$(I) = LEFTT$(L$ + "□□□□
     □□□□□□□□□□□□
     □□□",S2)
2170 NEXT
```

The ↑ key backs up if an incorrect match character is selected.

Unless labels are to be printed, this completes the setup process. For labels, one more question is asked: "Print a Dummy Label (y/n)". If the answer is "y", a sample label is printed and the question asked again. This allows the user to align the labels in the printer.

```
2180 IF PT$ < > "L" THEN 2330
2190 Z$ = " "
2200 FOR I = 1 TO NU
2210 : Z$ = Z$ + "XXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXX    "
2220 NEXT
2230 GOSUB 3510:REM TITLE
2240 PRINT "PRINT A DUMMY LABEL
     (Y/N)
2250 GOSUB 3470:REM GET
2260 IF WK$ < > "N" AND WK$
     "Y" THEN 2230
2270 IF WK$ = "N" THEN 2330
2280 FOR I = 1 TO 5
2290 : PRINT#4,Z$
2300 NEXT
2310 PRINT#4
2320 GOTO 2230
```

Z$ if concatenated to itself in lines 2200-2220 to make wider sample data for 2-up and 3-up printouts.

Printing

The main loop of the printout is from lines 2340-2630. It reads each record, sees if it matches code criteria and whether or not it fits "member only" and "local only" options. After all this, the record is printed *via* a GOSUB set in PT$. When all full-width lines have been completed, one more pass is made to do any less-than-full-width lines that remain; i.e., one last label in a 3-up output).

```
2340 FOR I = SN TO NV
...
2400 : GOSUB 4960:REM CHECK CODE
2410 : IF RC > 0 THEN 2450:
     REM FLUNKED CODE
2420 : IF MO = 0 THEN 2470: REM
     NOT FOR MEMBERS ONLY
2430 : WK$ = LEFT$(D1$(UC*NF),1)
2440 : IF WK$ = MM$ OR WK$ = MC$
     OR WK$ = MD$ THEN 2470
2450 : UC = UC - 1
2460 : GOTO 2630:REM SKIP NON-
     LOCALS
2470 : IF LO = 0 THEN 2520:REM NOT
     FOR LOCALS ONLY
2480 : WK$ = MID$(D1$(UC*NF),LP,1)
2490 : IF WK$ > = LL$ AND
     WK$ < = LH$ THEN 2520:
     REM LOCAL
2500 : UC = UC - 1
2510 : GOTO 2630: REM SKIP
...
2550 : TN = TN + NU
2560 : IF PT$ = "S" THEN GOSUB
     4170:UC = 0:REM SUMMARY
2570 : IF PT$ = "L" THEN GOSUB
     4520:UC = 0:REM LABELS
2580 : IF PT$ = "P" THEN GOSUB
     4690:UC = 0:REM PHONE LIST
2590 : IF PT$ = "V" THEN GOSUB
     4830:UC = 0:REM VISIT LIST
2600 : IF PT$ = "O" THEN GOSUB
     5110:UC = 0:REM SEQ. OUTPUT
2610 : IF PT$ = "N" THEN UC = 0
2620 : GET WK$:IF WK$ > " " OR
     FL THEN GOSUB 2780:REM
     PRINTER STATUS
2630 NEXT I
```

Variable TN keeps track of the total number printed thus far, adding NU, the number of labels in each row, to the total after each pass through the loop.

If a key is pressed during printing, or variable FL is non-zero, line 2620 branches to a subroutine that offers further options. After printing the record number(s) last printed, it waits for another key to be pressed. If that is a ← key, FL is set to halt the printer after each label, which is useful for cut sheets, loose cards, and individual envelopes. If the ↑ key is hit, FL is reset to restore non-stop printing. If the home key is hit, the printout is aborted safely by setting the record counter to the last record in the file.

```
2780 SW = I - NU
2790 IF PZ$ < > "N" THEN GOSUB
```

```
     3510:REM TITLE
2800 PRINT "RECORD(S) LAST PRINTED
2810 FOR WK = 1 TO NU
2820 : PRINT "□□□□"SW + WK;
2830 NEXT
2840 PRINT
2850 PRINT "PRESS ANY KEY TO
     CONTINUE
2860 GOSUB 3470:REM GET A KEY
2870 IF WK$ = " ← " THEN FL = 1:
     REM FAST LIST OFF
2880 IF WK$ = " ↑ " THEN FL = 0:
     REM FAST LIST ON
2890 I = SW + NU
2900 IF WK$ = "[home]" THEN I = NV:
     REM ABORT
2910 RETURN
```

Lines 2780 and 2890 adjust the current record count to allow the currently printed record numbers to be displayed.

This completes my discussion of the print module. Now I will take a detailed look at how a mail label is formatted and printed.

The output example is the common mail label and illustrates a way to preserve order in printouts whether one, two, or three labels wide. As mentioned above, it does double-duty, printing either straight mail labels or mail and phone number labels, depending on the contents of variable ZT (Zip Telephone).

The actual relative file data to be printed is read into the D1$( ) array by other routines similar to those studied in the update module (MICRO 58:85).

Here is the first part of the label routine:

```
4510 REM ** PRINT LABELS **
4520 IF ZT = 0 THEN 4570: REM
     NO PHONE OPTION
4530 D1$(4) = D1$(4) + "   " + D1$(PC)
4540 D1$(PC) = D1$(PH)
```

If ZT is non-zero in lines 4520-4540, the contents of the postal code field (PC) are appended to the contents of the last address field, and the phone field (PH) then takes the place of the postal code in the label. The same is done for a second and/or third record if more than one is to be printed at once.

```
4550 IFUC > 1 THEN D1$(4 + NF) =
     D1$(4 + NF) + "   " + D1$
     (PC + NF):D1$(PC + NF) = D1$
     (PH + NF)
4560 IFUC > 2 THEN D1$(4 + 2*NF) =
     D1$(4 + 2*NF) + "   " + D1$
     (PC + 2*NF):D1$(PC + 2*NF) =
     D1$(PH + 2*NF)
```

The D1$( ) array can hold three records at once, using NF (Number of Fields per record) as a step value between related elements of different records. This is more clearly seen in the actual printing routine:

```
4570 FOR II = 1 TO 5
4580 : FOR JJ = II TO II + UC*NF-NF
       STEP NF
```

Variable II counts the current line on the label, and JJ offsets to the current record horizontally. Since all fields of up to three records are part of the same array, JJ takes its value from II, offset by NF.

Outputs to CBM printers begin with a cursor down key contained in DN$.

```
4590 : : IF PZ$ = "P" THEN PRINT#4,
         DN$;
```

Outputs to ASCII printers begin with a SYS call that converts PETSCII to ASCII.

```
4600 : : IF PZ$ = "A" THEN SYS SM,
         1,D1$(JJ)
```

Line 4610 prints the field:

```
4610 : : PRINT#4,D1$(JJ)LEFT$(BL$,
         LW-LEN(D1$(JJ) ) );
```

After the field data is printed, blanks are printed to move the print head to the proper place for the same field of the next record to be printed. This is what allows the multi-column printouts.

Since each field printout ends with a semi-colon (";"), an extra PRINT statement ends the line after the JJ loop, and another adds the blank line between labels after the II loop.

If the user has requested the index card option (IC = 1) during setup, line 4660 adds extra lines between labels to move to the same spot on the next card:

```
4660 IF NU = 1 AND IC = 1 THEN FOR
     I1 = 1 TO 18:PRINT#4:NEXT
```

The other printout routines operate similarly and are largely copies of the above. Those in need of other special reports may add them as subroutines called from the main print loop beginning in line 2340.

One other routine in the print module is worthy of note. It serves two functions: first it expands Canadian postal codes to include a space after the first three characters at print time. (This same approach could be used with U.S. nine-digit zip codes and with telephone numbers.) It saves space on the disk, but takes longer to format for printing.

```
4970 P = UC*NF - (NF-PC)
4980 CD$ = D1$(P) + "            "
4990 AA = ASC(CD$)
5000 IF AA > 47 AND AA < 58
     THEN 5020
5010 D1$(P) = LEFT$(CD$,3) + "   " +
     MID$(CD$,4,3)
```

Line 4970 locates the postal code field of the current record. The next three lines determine whether or not the postal code is Canadian, assuming it will be if the third character is not a number. Then line 5010 does the actual insertion of the added space.

The second purpose of this routine is to see whether or not the current record meets any code match criteria that have been established. When CF is non-zero, a code search is made of each of the CF possible code field matches to see if any fit the current record. When either no code is wanted, or the right code is found, variable RC is cleared to zero; otherwise it is set to one and the calling routine will skip that record on printouts.

```
4960 RC = 1
...
5020 IF CF = 0 THEN RC = 0:GOTO 5090
5030 P = UC*NF
5040 FOR 12 = 1 TO CF
5050 : IF CF$(12) = MID$(D1$(P)
       + "              ",S1,S2) THEN RC = 0
5050 : IF CF$(12) = MID$(D1$(P)
       + "              ",S1,S2) THEN RC = 0
5060 : IF 12 > 1 OR CF$(1) <>
       MA$ THEN 5080
5070 : IF MID$(D1$(P) + "□□□□□
       □□□□□□□□□□□□
       □ □"
       ,S1,S2) > B5$ THEN RC = 0
5080 NEXT
5090 RETURN
```

Variable MA$ in line 5060 is the match-all character.

In the next and final installment of this series I'll describe the machine-language routines. Included will be a full source listing that will assemble for the PET, VIC-20, or Commodore 64. Conversion of the routines for the C64 has taken a little longer than expected, so the final installment will appear in the August issue.

Jim Strasma is assistant professor of computer science at Lincoln College. You may contact him at 1280 Richland Ave., Lincoln, IL 62656.

MICRO™

Design & Illustration: Paul Stephenson

ADVENTURE.
THE KEY IS YOUR COMPUTER

KONGO KONG

Climb ladders, avoid the barrels the crazy ape is rolling at you, and rescue the damsel. Fast machine code action.

Available for VIC-20 and COMMODORE 64

PLAYED WITH JOYSTICK OR KEYBOARD

$19.95

GRAVE ROBBERS

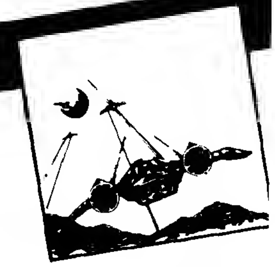
Introducing the first GRAPHIC ADVENTURE ever available on the VIC-20! With realistic audio-visual effects, you explore an old deserted graveyard and actually see the perils that lie beyond.

Available for VIC-20 and COMMODORE 64

PLAYED WITH KEYBOARD

$14.95

ANNIHILATOR



Defend your planet against the hostile aliens. All machine code makes this "Defender-like" program one of our best arcade games.

Available for VIC-20 and COMMODORE 64

PLAYED WITH JOYSTICK

$19.95

ADVENTURE PACK I

(3 programs)

Jack and the Beanstalk Defeat the Giant in your quest for his Golden Hen.

Computer Adventure Re-live the "excitement" of getting your computer. An adventure with a very different flavor.

Moon Base Alpha You must find a way to destroy the meteor that is racing towards your base, or else all moon colonies will be demolished!

Available for VIC-20 and COMMODORE 64

PLAYED WITH KEYBOARD

$19.95

ADVENTURE PACK II

(3 programs)

African Escape As the sole survivor of a plane crash, you must find your way out of the dark continent.

Hospital Adventure You are a spy whose mission is to complete the bungled assassination attempt on the evil dictator, who is recuperating in the hospital under heavy guard.

Bomb Threat Get back to town to warn the authorities of the bomb planted by the terrorists who left you prisoner at their hideout.

Available for VIC 20 and COMMODORE 64

PLAYED WITH KEYBOARD

$19.95

CHOMPER MAN

High speed machine action. Don't let the bullies catch you in a game packed full of machine code.

Available for COMMODORE 64

PLAYED WITH JOYSTICK OR KEYBOARD

$19.95

Programs for the VIC-20 and the COMMODORE

VICTORY SOFTWARE®

All programs fit standard memo All programs a on cassette ta

VICTORY SOFTWARE INC.
7 Valley Brook Road, Paoli, PA 19301
(215) 296-3787

Check your local dealer or order directly.

ORDERING We accept personal checks, money orders, VISA, and MASTER-CHARGE. Charge orders please include number and expiration date. Overseas orders please use charge or have check payable through a U.S. bank. Add $1.50 postage and handling per order. PA residents please add 6% sales tax.

Circle No. 39

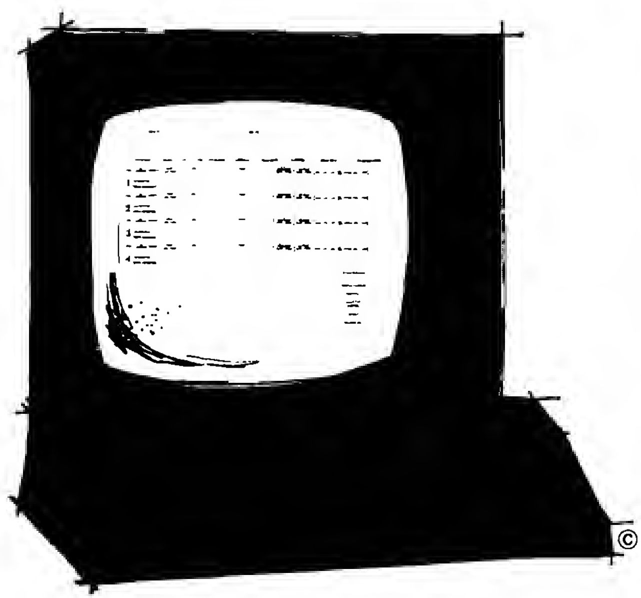
The MICRO™ LEARNING CENTER

Atari 800 **Texas Instrument**

Features:

- **Breakup for the VIC-20**
  by Loren Wright and Phil Daley
  A simple animation demonstration of PEEKing and POKEing screen memory.

- **Addressfile**
  by Brian Zupke
  A file management system for the VIC-20 and other Commodore computers



BREAK UP

by Loren Wright and Phil Daley

BREAKUP
requires:
VIC-20
Easily modifiable for other PETs

A simple animation demonstration of PEEKing and POKEing screen memory.

Get out your paddles or joystick and get ready to hit the ball with your bumper. By knocking down a wall of bricks, you can now show all your friends how dexterous you are.

Breakup is a simple animation display game that presents the principles of PEEKing and POKEing the screen memory to plot and erase characters on the screen, and to test for collisions. It includes a "ball" that moves around the screen, rebounds from struck objects, and knocks out blocks in the wall of bricks. It also includes a player-controlled "bumper" to keep the ball from going out-of-bounds and being lost, a defined playing field with three walls from which to bounce the ball, and two rows of blocks to be knocked off by your bouncing ball.

The game keeps score; 5 for each block and 1 for each bounce off the bumper. The previous highest score during each session is kept and displayed at the end of the game. More than 170 points qualifies you for a bonus of 50 points, new blocks, and an extra ball. You can choose to play with a joystick, paddles, or keyboard. If you choose the keyboard, the Function keys F1 and F7 move the bumper up and down. If you hold down the shift key at the same time, it increases the bumper movement speed.

Operating Instructions

1. Type in the BREAKUP program and type 'RUN'.
2. You are first asked whether you have a joystick, paddles, or will play from the keyboard. Choose the correct response — J, P, or K.
3. If you specify hardware that you don't have, you will have to press the RUN/STOP key to recover.
4. The program will display the playing field, the brick wall, and your bumper. When you are ready to start play, press the button on the joystick or paddle X, or the space bar if you are using the keyboard.

The Program

The ball starts from a random position on the left side of the screen and travels to the right, hitting a block. This causes the block to disappear, adds five to your score, and the ball rebounds toward the left. Here is the challenge: You must position the bumper to keep the ball from travelling out-of-bounds and off the screen, thereby losing the ball. If you are successful, the ball will rebound back toward the blocks. If you miss the ball, the game freezes so that you can see how close (or how far) you were from the ball when you missed it. You are allowed three balls (four if you get the bonus), so try not to miss. You will notice that the angle and speed of the ball increase the closer you hit the ball to the ends of the bumper. Hitting the ball near the center of the

bumper helps to restore the ball to the original direction.

The program stores the high score from game to game until the computer is turned off. Challenge your friends to a match and see who can get the highest score.

To create animation on a computer screen, we need to know how the computer displays information on the screen and how we can change that information. The television is constantly updating the information being displayed, about 30 times per second. A computer that uses memory-mapped video stores the screen information in RAM (Random Access Memory) so that the video display routine can have this information available at all times. The location in memory where this information is stored on the VIC depends on the amount of memory available on your particular VIC. The formula for determining this location is:

```
1. S = 4*(PEEK(36866)AND 128) +
   64*(PEEK(36869)AND112)
```

S returns with the start of video display memory location.

To plot a character in a given location on the screen, we must use the following equation:

```
2. A = S + X + 22 * Y
```

where A is the location in memory to either PEEK or POKE the character, depending upon the program usage. X and Y are the Cartesian coordinates of the location, starting at 0,0 in the upper left corner of the screen.

Appendix D of the *VIC Programmer's Reference Guide* contains a table of the codes to be POKEd in screen memory and the resulting characters. As an example, type equation 1 above into your VIC to assign the proper value to S, and then type 'POKE 36879,8 : POKE S, 81'. A small white ball should appear on your screen in the upper left corner. We used equation 2 above with X = 0 and Y = 0. By changing the X and Y values in that equation and POKEing the resulting locations with 81, you can print little white balls anywhere on the screen. X values can range from 0 to 22 and Y values from 0 to 20.

How do you erase them? Look up 'space' (which is 32 in the screen code chart and POKE that number into the same location as you originally plotted the ball. This effectively erases whatever is on the screen position. Location 36879 is the memory position for the background and border colors on the screen. Appendix B of the *VIC Programmer's Reference Guide* lists all the possible backgrounds and borders and the correct number to be POKEd into this location. The reason for turning the background black and plotting the ball white is that white is the default color when POKEing the screen memory locations.

POKEing an 81 to the S location with the screen set to white results in an invisible ball being plotted. To see the ball on a white screen we have to change the color of the ball. Each screen memory location also has associated with it a screen color memory location. For example, if you POKE 38520,2, this will change the white ball to red. Since two locations must be POKEd for any color other than white, we use white as the ball color to improve the speed of the program and lessen the complexity of the plotting routines.

Paddles and Joysticks

Using paddles or a joystick helps to make the game more interesting and easier to play, but programming them is a little obscure, especially for the novice BASIC programmer. If you don't understand the following information on first reading, look at the BREAKUP program and see how it is done, then reread this section.

There are five switches on a joystick. Four of them are controlled by the joystick and one is for the fire button. You must PEEK the locations of the VIA to determine which joystick switches are closed. The values are normally 0 and change to 1 (or −1) when closed. We also have to POKE the Data Direction Register to input mode. For VIA #1 this is accomplished easily by a POKE 37139,0. Changing VIA #2 to input disables keyboard input, so we change only the DDR when we are polling the joystick and return it to normal during the rest of the program. Therefore, we POKE 37154,127, read the joystick, and POKE 37154,255 immediately afterward.

Reading switches 0, 1, 2, and 4 are done as follows:

```
POKE 37139,0
S0 = ((PEEK(37137)AND4)=0)
S1 = -((PEEK(37137)AND8)=0)
S2 = ((PEEK(37137)AND16)=0)
S4 = -((PEEK(37137)AND32)=0)
```

Reading switch 3 is a bit trickier:

```
POKE 37154,127
S3 = -((PEEK(37152)AND128)=0)
POKE 37154,255
```

Because BREAKUP uses only up and down readings of the joystick, the switch 3 program lines are not needed. Switch 0 is North, switch 1 is South, switch 2 is West, switch 3 is East, and switch 4 is the fire button.

The VIC paddles are read in the same manner except that an actual value may be PEEKed for the paddle settings. The switches are the same. Here are the lines necessary to implement paddle readings:

```
POKE 37139,0                  Set DDR to Input
PX = PEEK(36872)              Paddle X value
SX = -((PEEK(37137)
 AND16=0)                     Paddle X switch
PY = PEEK(36873)              Paddle Y value

POKE 37154,127
SY = -((PEEK(37152)
 AND128)=0)
POKE 37154,255                Paddle Y switch
```

Program Description

Lines 1000-1020 (the subroutine to plot the bumper) calculate the screen position of the bumper and then POKE the locations either with a bar (B = 245) or a space (B = 32), depending on whether the subroutine is called to plot or erase the bumper. Lines 3070 and 3090 erase and draw the ball, respectively. Line 3080 looks at the screen location to see if a block has been hit. If so, the subroutine at line 5000 is called to update the score and put "english" on the ball if it was called because of bumper contact.

Lines 4050-4060 plot the block walls. If you remove lines 4050 and 4060, the game turns into "PONG" with the player scoring one point for each rebound from the bumper. Line 4140 picks a random spot along the left-hand side of the screen for the new ball to appear.

Line 3050 tests for out-of-bounds in the Y range and line 3100 tests for out-of-bounds in the X range. If the ball is out-of-bounds it reverses the ball direction. Line 3120 tests for hitting the bumper. If it has hit the bumper then the 5000 subroutine is called to update the score and put english on the ball, as previously mentioned.

As a whole, the routine moves rather slowly due to the limitations of BASIC language. We employed several techniques to keep the program moving as fast a possible. Variables used most often (those concerning the ball and bumper) were initialized at the beginning of the program so that they would be placed in the beginning of the variable storage area. The screen update subroutines were placed near the beginning of the program so that the interpreter would spend less time looking for the line numbers of the subroutines. In line 2010 (the bumper moving routine) a return is placed in the first line so that when the bumper is not to be moved, the program doesn't spend time replotting an unmoved bumper. That is why the ball moves much faster when you are not changing the bumper location on the screen.

The screen POKEs can be any character (see the screen code chart in the programming manual). Breakup uses 32 (the space) to erase the screen. The ball is 81, the blocks are 102, the bumper is 245. These could be changed easily to different shapes. The scoring subroutine at line 5000 can be changed to make the score advance either faster or slower. Also, the scoring could be changed to increase the bonuses or provide snappy comments at the conclusion of the game. The block-plotting routine at line 4050 could be modified to increase or decrease the number of columns of blocks. If you plot them nearer to the bumper, then the 15 in line 3080 will need to be decreased commensurately.

Changes for Other PETs

The POKEs in 3050, 3080, 3110, 3130, 3150, 4020, 4170, 4250, 6000, and 6090 may be removed. They concern screen color and bounce sounds.

```
4000 COL=40
4010 S0=32768
```

After typing the program as is, other PET owners should make these changes:

```
2090 S=0:IFA$=" = "THENS=1
2100 IFA$=CHR$(20)THENS= -1
2110 IFA$=" ∧ "THENS=3
2120 IFA$=CHR$(148)THENS= -3
```

Line 4010 is also different for the C-64:

```
4010 S0=1024
```

You may contact the authors at MICRO Magazine, Box 6502, Amherst, New Hampshire 03031.

Breakup Listing

```
10 X=0:Y=X:X1=X:Y1=X:S0=X:COL=X:S1=X:S2=S:S3=X:DY=X:DX=X:GOTO4000
1000 S1=S0+COL*A:S2=S0+COL*(A+1):S3=S0+COL*(A+2)
1010 POKES1,B:POKES2,B:POKES3.B
1020 RETURN
2000 TS=INT(PEEK(36872)/17)+2
2010 IFTS<A+2ANDTS>A-2THENRETURN
2020 B=SP:GOSUB1000
2030 A=TS:GOTO2150
2040 Q0=-((PEEK(PA)AND4)=0):Q1=((PEEK(PA)AND8)=0)
2050 IFQ0<>0THENS=-3:GOTO2130
2060 IFQ1<>0THENS=3:GOTO2130
2070 RETURN
2080 GETA$:IFA$=""THENRETURN
2090 S=0:IFA$="■"THENS=1
2100 IFA$="■"THENS=-1
2110 IFA$="■"THENS=3
2120 IFA$="■"THENS=-3
2130 B=SP:GOSUB1000
2140 A=A+S
2150 IFA<2THENA=2
2160 IFA>17THENA=17
2170 B=D:GOSUB1000
2180 RETURN
3000 OK=1
3010 ONPDLGOSUB2000,2040,2080
3020 X1=X:Y1=Y
3030 Y=Y+DY:IFY>19THENY=19
3040 IFY<2THENY=2
3050 IFY=2ORY=19THENDY=-DY:POKEV2,230
3060 X=X+DX:IFX>19THENX=19
```

(Continued on page 78)

Addressfile

by Brian Zupke

The program Addressfile stores names, addresses, phone numbers, and memos in cassette files. Entries can be entered, deleted, or changed, and files can be updated at any time. By using the search routine, you can locate any entry having a particular name, address, phone number, or memo.

The program was written on a 13K VIC-20. However, if the REM statements are deleted, it will run with 8K. Since the program is somewhat large, I am offering to make a copy of it for anyone who sends me either a blank tape and a self-addressed, stamped envelope, or $3.00.

To allow for fast and easy data entry and editing, INPUT statements were used. This means that no colons or commas are allowed. The program has eight major functions: ADD ENTRIES, CHANGE ENTRIES, VIEW ENTRIES, DELETE ENTRY, SEARCH FILE, SAVE FILE, LOAD FILE, and END PROGRAM. Below are detailed descriptions of each function.

ADD ENTRIES — Add new entries to the file.

Added entries are put at the end of the current file, or the file presently in memory. After typing an entry, the computer asks if you want to make any changes. The default is 'N'. If you type 'Y', then the cursor moves to the top of the entry and it is re-entered. If you don't want to change a certain line in the entry, just hit RETURN; otherwise make the change and then hit RETURN. Be sure to hit RETURN on each line of the entry since the computer is INPUTing all of them again. If you do not want to make any changes then the computer will ask if you are finished adding entries to the file. The default is 'N'. You must type a 'Y' to return to the menu.

CHANGE ENTRY — Change any part or all of an entry.

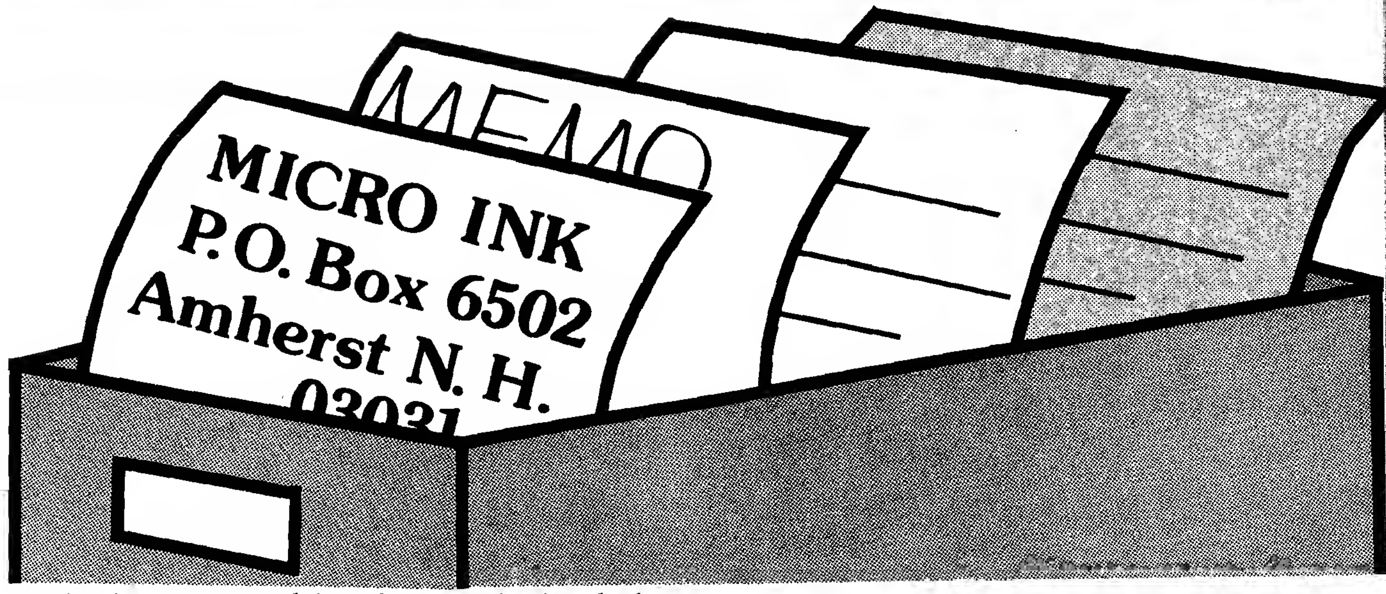
The computer first asks for the number of the entry to be changed. If you don't want to change an entry respond with a number less than one, which returns control to the menu. When a valid entry number is typed in, that entry is listed. Question marks are printed out before each line so that the entry can be re-entered. Change the entry as you would when adding entries. After the changes are made, the computer asks if you want to change another entry. The default is 'Y'. Control is returned to the menu if you type 'N'.

VIEW ENTRIES — View any or all entries in the file.

The computer first asks for a starting entry to be viewed. When a number is typed in, the corresponding entry is displayed. The computer will then ask if you want to see another entry. The default is 'Y'. If the response is 'N', then control is returned to the menu; otherwise, the entry immediately following the one on the screen is displayed. For example, if the sixth entry is on the screen, the seventh entry will be displayed next. After the last entry is displayed and you request to see another entry, the computer will display an end-of-file message briefly and control is returned to the menu.

DELETE ENTRY — Delete an entry in the file.

The computer asks for the number of the entry to be deleted. Once a number is typed in, the corresponding entry is displayed. The computer



asks if you want to delete the entry displayed. If you type 'Y', then the entry is deleted and all the entries above that one are moved down one. For example, if entry six is deleted, then entry seven becomes entry six, eight becomes seven, etc. If you type 'N' the entry is not deleted. In either case, the computer will ask for another entry number. If you type a number less than one, control is returned to the menu.

SEARCH FILE — Search the current file for a specific name, address, etc.

There are four things you can search for in the file: the name, address, phone number, or memo. After choosing one, the computer will ask if you want a full printout of each entry in the file that matches. The default is 'Y'. If you type 'N' then only the entry number is printed. Then the computer will ask if you want a pause after each match is printed. Again the default is 'Y;. With the pause, a key will have to be pressed after each match is listed. Then the computer will ask for the specific item for which to search. If you are searching for a name and two names are typed in (first and last), the computer will check both names individually. This is true for names in an entry. If either name typed in matches either name in an entry, then a match is found. When searching for an address, a match is found if the street is the same, the city is the same, or if the state and zip code are the same as the entry's. With the phone number and memo, a match is found when they are the same as the entry's. When all entries have been checked an end of search message is displayed. A key has to be pressed to return to the search menu. If you are through searching, press '5' to return to the menu.

SAVE FILE — Save current file on cassette tape.

The computer asks for a name under which to save the file. After you type in the name, the computer will display the press play and record message. Once you start the tape, the computer will open and save the file, and once the file is saved, it is CLOSED and control is returned to the menu. When you save a file you must be sure of two things: don't overwrite another file, and make sure that each line in every entry has at least one character or number in it. If you fail to do this the entries will contain the wrong information when the file is loaded back in. This happens because the computer will pass over a blank line and INPUT# the first non-blank line in the same variable, which causes the entries to 'bunch up.'

LOAD FILE — Load a file from tape into the computer.

The computer asks for the name of the file you wish to load. When a name is typed in, the corresponding file is loaded into the computer. Control is returned to the menu after the file is loaded.

END PROGRAM — Terminates program.

Caution should be taken with this function. Be sure that you save any wanted files on tape before quitting.

(Addressfile Listing on next page)

Addressfile Listing

```
10 REM      ***************
11 REM      *             *
12 REM      * ADDRESSFILE *
13 REM      *             *
16 REM      *     BY      *
17 REM      *             *
18 REM      * BRIAN ZUPKE *
19 REM      *             *
20 REM      ***************
99 REM
100 DIMN$(100),S$(100),C$(100),T$(100),Z$(100),M$(100),P$(100):E=1:Q$="? "
130 REM * PRINT MAIN MENU *
150 PRINT"     ADDRESSFILE"
160 PRINT"";SPC(8)"MENU":PRINT"  1) ADD ENTRIES":PRINT"  2) CHANGE ENTRIES
"
170 PRINT"  3) VIEW ENTRIES":PRINT"  4) DELETE ENTRY":PRINT"  5) SEARCH FILE"
175 PRINT"  6) SAVE FILE":PRINT"  7) LOAD FILE":PRINT"  8) END PROGRAM"
177 PEM INPUT USERS CHOICE
180 PPINT"       -CHOICE-":INPUT"      ";C
182 IFC=8THENEND
185 REM JUMP TO CORRECT ROUTINE
190 ONCGOTO200,300,400,700,800,500,600
191 GOTO150
195 REM * ADD ENTRY TO FILE *
200 PPINT"      ADD ENTPIES"
210 PPINT"ENTRY #"E""
220 PRINT"NAME:":INPUTN$(E)
230 PRINT"STREET:":INPUTS$(E)
240 PRINT"CITY:":INPUTC$(E)
250 PRINT"STATE:":INPUTT$(E)
260 PRINT"ZIP-CODE:":INPUTZ$(E)
265 PRINT"PHONE:":INPUTP$(E)
270 PRINT"MEMO:":INPUTM$(E)
275 REM CHECK IF NEW ENTRY NEEDS TO BE CHANGED
280 INPUT"CHANGESN";Y$:IFLEFT$(Y$,1)<>"Y"THEN294
285 REM IF CHANGE THEN CURSOP UP & INPUT ENTRY AGAIN
290 PRINT"":GOTO220
293 REM CHECK IF DONE ADDING ENTPIES
294 E=E+1:INPUT"DONEN";Y$:IFLEFT$(Y$,1)="Y"THEN150
295 GOTO 200
297 REM * CHANGE ENTRY IN FILE *
299 REM INPUT ENTRY # TO BE CHANGED
300 PRINT" CHANGE ENTRY":INPUT"ENTRY #";X:PRINT"";Q$;N$(X):PRINTQ$;S$(X)
305 REM DISPLAY ENTRY
310 PRINTQ$;C$(X):PRINTQ$;T$(X):PRINTQ$;Z$(X):PRINTQ$;P$(X):PRINTQ$;M$(X)
315 PEM CUPSOR UP & ENTER ENTRY
320 PRINT"":INPUTN$(X):INPUTS$(X):INPUTC$(X)
321 INPUTT$(X):INPUTZ$(X):INPUTP$(X):INPUTM$(X)
325 PEM CHECK FOP MORE ENTPY CHANGES
330 INPUT"ANY MOPEY";Y$:IFLEFT$(Y$,1)="Y"THEN300
340 GOTO150
360 REM * VIEW ENTPIES IN FILE *
370 REM INRUT STARTING ENTRY
400 PRINT"  VIEW ENTRIES":INPUT"STARTING ENTRY";Y:PRINT""
410 FORX=YTOE-1:PRINTN$(X):PPINTS$(X):PRINTC$(X)
415 PRINTT$(X),Z$(X):PRINTP$(X):PRINTM$(X)""
420 PRINT"ENTRY #";X:INPUT"CONTINUE    Y";Y$:IFLEFT$(Y$,1)<>"Y"THEN150
425 REM CONTINUE VIEW
427 REM PRINT END OF FILE MESSAGE WHEN DONE
430 PRINT"":NEXTX:PRINT"END OF FILE":FORX=1TO2000:NEXT:GOTO150
480 REM * SAVE FILE ON TAPE *
500 PRINT"  SAVE FILE"
505 REM INPUT NAME FOR FILE
510 PRINT"FILENAME:":INPUTA$
512 IFA$="STOP"THEN150
515 PEM SAVE ALL ENTRIES IN NAMED FILE
520 OPEN1,1,1,A$
530 PRINT#1,E:FORX=1TOE-1
540 PRINT#1,N$(X):PRINT#1,S$(X):PPINT#1,C$(X)
543 PRINT#1,T$(X):PRINT#1,Z$(X):PRINT#1,P$(X):PRINT#1,M$(X)
545 REM END SAVE
550 NEXT:CLOSE1:GOTO150
580 REM * LOAD FILE FROM TAPE *
600 PRINT"LOAD FILE"
605 REM INPUT NAME OF FILE TO LOAD
610 PRINT"FILENAME:":INPUTA$
612 IFA$="STOP"THEN150
615 REM LOAD NAMED FILE FROM TAPE
620 OPEN1,1,0,A$
630 INPUT#1,E:FORX=1TOE-1
640 INPUT#1,N$(X):INPUT#1,S$(X):INPUT#1,C$(X)
643 INPUT#1,T$(X):INPUT#1,Z$(X):INPUT#1,P$(X):INPUT#1,M$(X)
645 PEM END OF LOAD
650 NEXTX:CLOSE1:GOTO150
680 REM * DELETE ENTRY FROM FILE *
700 PRINT"  DELET ENTFY"
701 REM INPUT ENTRY # TO BE DELETED
705 PRINT"ENTRY #:":INPUTN
710 IFN<1THEN150
715 PEM IF # IS OUT OF BOUNDS, STOP
720 IFN>E-1THENPRINT"ENTPY DOES NOT EXIST":GOTO705
723 PEM FRINT ENTPY
725 PRINT"";N$(N):PRINTS$(N):PRINTC$(N):PRINTT$(N),Z$(N):PRINTP$(N):PRINTM$(N)
;""
727 PEM LAST CHANCE NOT TO DELETE ENTRY
730 INPUT"DELETE";Y$:IFLEFT$(Y$,1)<>"Y"THEN700
735 PEM MOVE ALL ENTRIES ABOVE # DOWN ONE
740 FORX=NTOE-1:N$(X)=N$(X+1):S$(X)=S$(X+1):C$(X)=C$(X+1)
743 T$(X)=T$(X+1):P$(X)=P$(X+1):M$(X)=M$(X+1)
745 Z$(X)=Z$(X+1)
747 PEM REDUCE TOTAL # OF ENTRIES
750 NEXT:E=E-1:GOTO700
780 REM * SEARCH ENTRIES IN FILE *
800 PRINT"  SEARCH FILE"
810 FRINT"  CHOICES:":PRINT"1) SEARCH FOR NAME"
820 FPINT"2) SEARCH FOR ADDPESS":PRINT"3) SEARCH FOR PHONE #"
825 FRINT"4) SEARCH FOP MEMO":PRINT"5) RETURN TO MENU"
830 INPUT"CHOICE";C:IFC<1ORC>5THEN800
840 IFC=5THEN150
842 PEM  CLEAR FLAGS
845 P=0:W=0
850 PRINT"DO YOU WANT A FULL    PRINTOUT OF EACH       POSSIBLE MATCH?"
856 REM FOR A PRINTOUT OF EACH MATCH SET PRINTOUT FLAG
860 INPUT"Y";Y$:IFLEFT$(Y$,1)="Y"THENP=1
870 PRINT"DO YOU WANT A PAUSE   AFTER EACH LISTING?"
876 PEM FOR PAUSE AFTER EACH MATCH SET PAUSE FLAG
880 INPUT"Y";Y$:IFLEFT$(Y$,1)="Y"THENW=1
886 REM JUMP TO CORPECT SEARCH INPUT
890 ONCGOTO900,940,980,970
895 REM * INPUT NAME TO SEARCH *
900 PPINT"NAME:":INPUTN$:IFN$="STOP"THEN800
910 FORX=1TOLEN(N$):IFMID$(N$,X,1)=" "THEN930
920 NEXT:L$=N$:R$="":GOTO1000
930 L$=MID$(N$,1,X-1):R$=MID$(N$,X+1):GOTO1000
936 PEM * INPUT ADDRESS TO SEARCH *
940 PPINT"USE STANDARD          ABREVIATIONS."
950 PRINT"STREET/P.O.BOX:":INPUTS$:IFS$="STOP"THEN800
960 PPINT"CITY:":INPUTC$:PPINT"STATE:":INPUTT$:PRINT"ZIP CODE:":INPUTZ$:GOTO100
0
966 REM * INPUT MEMO TO SEARCH *
970 PPINT"MEMO:":INPUTM$:IFM$="STOP"THEN800
972 GOTO1000
976 REM * INPUT PHONE # TO SEARCH *
980 PRINT"PHONE #:":INPUTP$:IFP$="STOP"THEN800
995 PEM * START OF SEARCH ROUTINE *
1000 PFINT"SEAPCHING FOR ENTRY"
1006 REM JUMP TO CORRECT MATCH CHECK
1010 FORX=1TOE-1:ONCGOTO1100,1150,1210,1190
1015 REM CONTINUE SEARCH
1020 NEXTX:PRINT"END OF SEARCH":PRINT"HIT KEY TO CONTINUE"
1030 GETA$:IFA$=""THEN1030
1035 GOTO800
1037 REM MATCH FOUND-PRINT ENTRY #
1040 PRINT"ENTRY #";X:IFP=0THEN1060
1046 REM PRINT NAME,ADDRESS & MEMO OF MATCH
1050 PRINTN$(X):PRINTS$(X):PRINTC$(X)
1055 PRINTT$(X),Z$(X):PRINTP$(X):PRINTM$(X)
1060 IFW=0THEN1020
1070 PRINT"HIT KEY TO CONTINUE"
1076 REM PAUSE BEFORE CONTINUING SEARCH
1080 GETA$:IFA$=""THEN1080
1090 GOTO1020
1094 REM * CHECK NAME OF ENTRY FOR MATCH *
1096 REM FIND FIRST & LAST NAME
1100 FORY=1TOLEN(N$(X)):IFMID$(N$(X),Y,1)=" "THEN1120
1110 NEXTY:F$=N$(X):S$="-":GOTO1135
1120 F$=MID$(N$(X),1,Y-1):S$=MID$(N$(X),Y+1)
1130 REM MATCH IF SAME FIRST OR LAST NAME
1135 IFL$=F$ORR$=S$ORL$=S$ORR$=F$THEN1040
1137 REM NO MATCH FOUND, CONTINUE SEARCH
1140 GOTO1020
1142 REM * CHECK ADDRESS OF ENTRY FOR MATCH *
1148 REM MATCH IF SAME STREET
1150 IFS$=S$(X)THEN1040
1155 REM MATCH IF SAME CITY
1160 IFC$=C$(X)THEN1040
1165 REM MATCH IF SAME STATE & ZIP CODE
1170 IFT$=T$(X)ANDZ$=Z$(X)THEN1040
1175 REM NO MATCH FOUND, CONTINUE SEAPCH
1180 GOTO1020
1182 REM * CHECK MEMO OF ENTRY FOR MATCH *
1188 MATCH IF SAME MEMO
1190 IFM$=M$(X)THEN1040
1195 REM NO MATCH FOUND, CONTINUE SEARCH
1200 GOTO1020
1207 REM * CHECK PHONE # OF ENTRY FOR MATCH *
1209 REM MATCH IF SAME PHONE #
1210 IFP$=P$(X)THEN1040
1215 REM NO MATCH FOUND, CONTINUE SEARCH
1220 GOTO1020
READY.
```

MICRO™

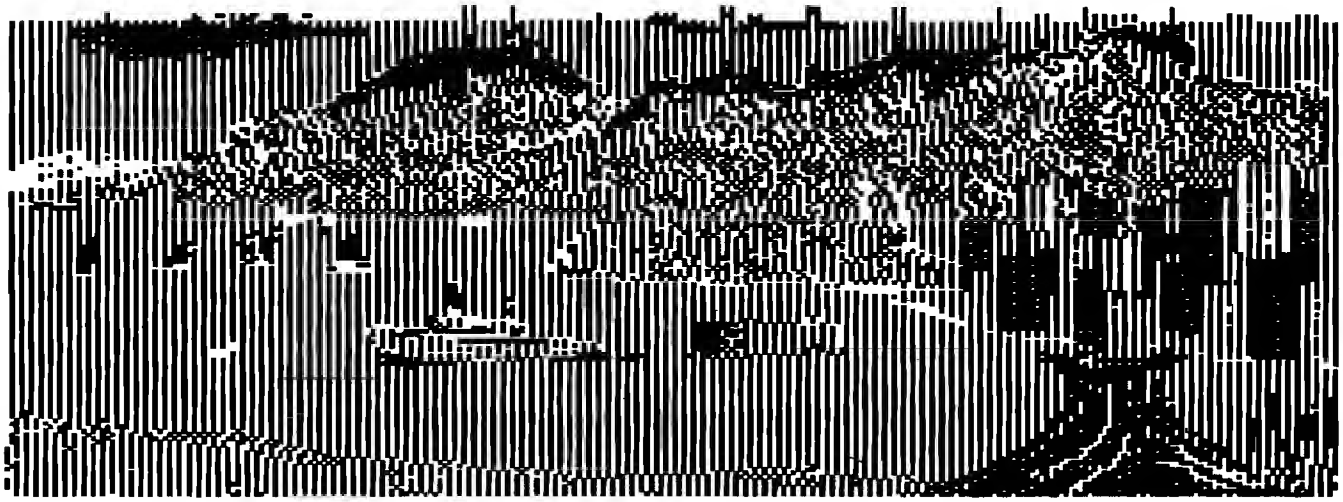
Breakup Listing *(continued)*

```
3070 POKES0+X1+COL*Y1,SP
3080 IFX>15ANDPEEK(S0+X+COL*Y)=102THENPOKEV1,255:Q=5:GOSUB5000
3090 POKES0+X+COL*Y,BL
3100 IFX>1ANDX<19THEN3150
3110 IFX=19THENDX=-DX:POKEV3,230:GOTO3150
3120 IFX=1AND(Y>=AORY=A-1ANDDY>0)AND(Y<=A+2ORY=A+3ANDDY<0)THENFLAG=1
3130 IFFLAG=1THENFLAG=0:DX=-DX:POKEV3,220:GOSUB5000:GOTO3150
3140 BALL=BALL-1:OK=0:POKES0+X1+COL*Y1,SP:RETURN
3150 POKEV1,0:POKEV3,0:RETURN
4000 V1=36874:V3=36876:CS=36879:PA=37137:POKE37139,0:COL=22
4010 S0=4*(PEEK(36866)AND128)+64*(PEEK(36869)AND112)
4020 POKECS,30:GOSUB6000:POKE650,128
4030 A=2:B=D:GOSUB1000
4040 BALL=3:Q=1
4050 FORX=16TO18STEP2:FORY=2TO19
4060 POKES0+X+COL*Y,102:NEXT:NEXT
4070 IFBNUSTHEN4090
4080 SCRE=-1:GOSUB5000
4090 ONPDLGOTO4100,4100,4130
4100 BUT=-((PEEK(PA)AND(PDL*16))=0)
4110 IFBUT<>1THEN4100
4120 GOTO4140
4130 GETA$:IFA$<>" "THEN4130
4140 X=2:Y=INT(RND(1)*14)+2:DX=1:DY=1
4150 GOSUB3000
4160 IFOKTHEN4150
4170 POKEV3,0
4180 ONPDLGOTO4190,4190,4220
4190 BUT=-((PEEK(PA)AND(PDL*16))=0)
4200 IFBUT<>1THEN4190
4210 GOTO4230
4220 GETA$:IFA$<>" "THEN4220
4230 POKES0+X+COL*Y,SP
4240 IFBALLTHENSCRE=SCRE-1:GOSUB5000:GOTO4140
4250 POKECS,27:PRINT"GAME OVER!
4260 PRINT"YOU SCORED"SCRE""
4270 PRINT"POINTS!
4280 PRINT"PREVIOUS HIGH WAS"HI""
4290 PRINT"PRESS SPACE TO CONT.":IFSCRE>HITHENHI=SCRE
4300 GETA$:IFA$=""THEN4300
4310 GOSUB6090
4320 GOTO4030
5000 SCRE=SCRE+Q:PRINTV$"BALL="BALL"  SCORE="SCRE;
5010 IFNOTBNUSANDSCRE>170THENSCRE=SCRE+50:BALL=BALL+1:BNUS=-1:POKEV1,0
     :GOTO4050
5020 IFQ=5THENDX=-DX:GOTO5080
5030 IFY<=AANDDY>0THENDY=-2*DY:GOTO5080
5040 IFY>=A+1ANDDY<0THENDY=-2*DY:GOTO5080
5050 DY=DY/2
5060 IFDY>0ANDDY<1THENDY=1
5070 IFDY<0ANDDY>-1THENDY=-1
5080 Q=1:RETURN
6000 POKE36878,15:PRINT""
6010 PRINT"VIC BREAKUP"
6020 PRINT"KEYBOARD, PADDLE":PRINT"JOYSTICK OR QUIT?"
6030 GETA$:IFA$=""THEN6030
6040 IFA$="P"THENPDL=1:GOTO6090
6050 IFA$="J"THENPDL=2:GOTO6090
6060 IFA$="K"THENPDL=3:GOTO6090
6070 IFA$="Q"THENPRINT"":END
6080 GOTO6030
6090 POKECS,8
6100 V$=""
6110 D=245:SP=32:BL=81
6120 PRINT""
6130 PRINT"   "
6140 FORI=1TO18:PRINTTAB(20);"":NEXT
6150 PRINT"   "
6160 RETURN
READY.
```

MICRO™

Apple Graphics for Okidata Microline 80: Revisited

by Gary Little



This article describes how to print the Apple II's high-resolution graphics screens on an Okidata Microline 80 printer.

Graphics Dump
requires:
Apple II and
Microline 80 printer.

I've had my Okidata Microline 80 printer for almost two years. When I first bought it, I used it primarily to facilitate program development. Currently, I am using it also for writing letters, reports, and magazine articles such as this one.

Until recently I had never used the printer for generating a hard-copy version of the display on either of the Apple's high-resolution screens. I didn't have a need to do this because I rarely write graphics software. Even so, I recognized that it would be a challenge to convert the pixels on an Apple graphics screen into a stream of bytes that would tell the Okidata to print one of its TRS-80-style 2×3 block-graphics characters, and still have the resultant output resemble the original picture!

The problem became more difficult when I discovered that to print a block-graphics character, the high-order bit of each byte sent to the printer has to be set to 1. The firmware on my Centronics-style parallel interface card sets this bit to 0. Then I read "Apple Graphics for Okidata Microline 80" by Charles F. Taylor (MICRO 48:48). That article described a Pascal program to solve the very problem I was attempting to solve. It also included a 6502 assembly-language program that could be used in a DOS environment. In addition, Charles described a simple method to circumvent the high-bit hardware problem.

A quick review of the article convinced me that there was still some programming to be done. I wanted to be able to print the *full* graphics screen sideways on the paper. This would give the largest size possible and would fit nicely on legal-size 8½" × 14" paper. The Pascal program listed in Charles's article does this, but not the assembly-language program. Since I don't use Pascal, I had to solve the 'full screen' problem using 6502 assembly language.

Solving the Hardware Problem

To solve the hardware problem previously mentioned, all you have to do is locate the wire used to transmit the high-order data bit between the printer interface card and the printer connector, sever it, and then extend the wire from the printer side to pin 15 of the game I/O connector. This pin corresponds to what is called annunciator 0 and can be set to a high or low voltage by reading locations $C059 or $C058, respectively. This, in turn, will either set or clear the high-order bit of the character being sent to the printer (at least in the printer's eyes).

From 6502 assembly language, enter the command

```
LDA $C059
```

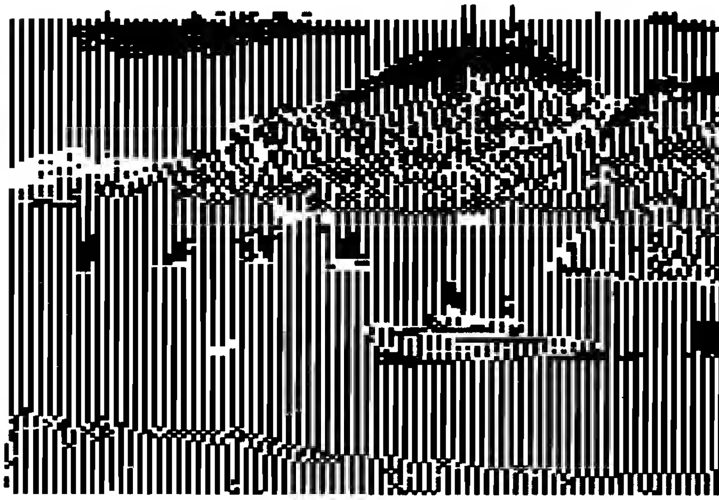
to set the high bit before sending graphics characters, or enter the command

```
LDA $C058
```

to clear the high bit before sending ordinary text.

That's it for the hardware problem. If this modification is made there is no need to change the hardware switch described in Charles's article since everything is handled under software control. And since annunciator 0 is off when the Apple is first turned on, or after a RESET, there is little danger that you will send graphics when you intend

Apple Graphics
Microline 80: R



This article describes how to print the Apple II's high-r graphics screens on an Okidata Microline 80 printer.

Graphics Dump requires:

Apple II and
Microline 80 printer.

I've had my Okidata Microline 80 printer for almost two years. When I first bought it, I used it primarily to facilitate program development. Currently, I am using it also for writing letters, reports, and magazine articles such as this one.

Until recently I had never used the printer for generating a hard-copy version of the display on either of the Apple's high-resolution screens. I didn't have a need to do this because I rarely write graphics software. Even so, I recognized that it would be a challenge to convert the pixels on an Apple graphics screen into a stream of bytes that would tell the Okidata to print one of its TRS-80-style 2×3 block-graphics characters, and still have the resultant output resemble the original picture!

The problem became more difficult when I discovered that to print a block-graphics character, the b each byte sent to the pı set to 1. The firmwar tronics-style parallel int this bit to 0. Then Graphics for Okidata M Charles F. Taylor (MICR article described a Pas solve the very problem ting to solve. It also ir assembly-language prog be used in a DOS enviro tion, Charles descrit method to circumven hardware problem.

A quick review of t vinced me that there programming to be done able to print the *full* g sideways on the paper. T the largest size possible nicely on legal-size 8½' The Pascal program list article does this, but no language program. Sinc Pascal, I had to solve t problem using 6502 asse

Solving the Hardware

To solve the hard

to send text to the printer. (If this happens, press RESET to re-enable 'text' mode.)

Solving the Software Problem

The MICRO article was most useful to me for its description of how to convert six pixels on the Apple screen into an equivalent TRS-80 2×3 block-graphic character. All I needed to write was a high-resolution SCRN subroutine to tell me whether a particular pixel was on or off. This subroutine begins at line 127 of the source listing shown below. It is a general-purpose subroutine and could be used in other applications, if required. Upon return from the subroutine, the carry flag will be set if the pixel corresponding to row ROWNUM and column COLNUM is on, and will be clear if the pixel is off.

Since the picture is going to be printed sideways on the paper, the rows of each 2×3 block-graphic character will correspond to columns on the high-resolution screen and the columns to rows. For a given ROW and COLUMN base, the corresponding block-graphic character is assembled as indicated in figure 1.

Figure 1: Okidata Microline 80's 2×3 Block-Graphics Characters

	ROW	ROW-1
COLUMN	b1	b2
		b7 = 0 and b8 = 1
COLUMN + 1	b3	b4
	data byte: b8b7b6b5b4b3b2b1	
COLUMN + 2	b5	b6

A sub-block is on if the corresponding pixel for (ROW,COLUMN) on the high-resolution screen is on, and is off if the pixel is off. The labels b1, b2, etc., refer to the bits in the block-graphic character byte.

The algorithm for printing the whole screen (page 2) is as follows:

1. ROW = 191 : COLUMN = 0
2. Assemble block graphic and print it
3. ROW = ROW − 2 : Repeat step 2 if ROW is not equal to $FF
4. ROW = 191 : COLUMN = COLUMN + 3 : Repeat step 2 if COLUMN 279

Page 1 of graphics can be dumped using the same routine by changing the variable PAGE from $40 to $20.

How to Use the Program

The program resides in memory beginning at location $6000, just above graphics page 2. (You will have to reassemble the source code if you wish to use the routine at a different location.) Use the BLOAD command to load it at this address from the diskette. Once your graphics image is in memory, enter the command CALL 24576 from BASIC to dump the image to the Microline 80. Note that the process can be aborted at any time by pressing ESC.

Although it takes several minutes for the full high-resolution screen to be printed, the results are worth it. Figure 1 shows an example of the result of a dump of graphics page 2.

Conclusion

The assembly-language routine presented here can be used within larger assembly-language programs or can be CALLed from an Applesoft or Integer BASIC program. The hardcopy output will be centered perfectly if legal-size paper is used. I suspect that this program, together with those presented in Charles Taylor's article, will complete your Microline 80 graphics dump software library.

Mr. Little is a lawyer in Vancouver specializing in computer-related law. He owns an Apple II and is currently Vice President of Apple's British Columbia Computer Society. You may contact Mr. Little at #214-131 Water St., Vancouver, B.C., Canada V6B 4M3.

***Listing 1:* Graphics Dump**

```
                  3  ************************
                  4  * HIGH-RES SCREEN DUMP *
                  5  *       for the        *
                  6  * OKIDATA MICROLINE 80 *
                  7  *                      *
                  8  *  by Gary B. Little   *
                  9  *     May 05/1982      *
                 10  ************************
                 11  GBASL    EPZ $26
                 12  GBASH    EPZ GBASL+$1
                 13  ESCHR    EQU $638
                 14  FLAGS    EQU $6B8
                 15  PWDTH    EQU $4B8
                 16  MODE     EQU $5B8
                 17  KBD      EQU $C000
                 18  STROBE   EQU $C010
                 19  HIGHON   EQU $C059    *Turns on Bit #7
                 20  HIGHOFF  EQU $C058    *Turns off Bit #7
                 21  PRINTGR  EQU $C102
                 22           ORG $6000
                 23  **********************
                 24  * Initialize printer *
                 25  **********************
6000 8D 58 C0    26           STA HIGHOFF  *Turn off Graphics
6003 A2 C1       27           LDX #$C1
6005 A9 09       28           LDA #9
6007 9D B8 06    29           STA FLAGS,X
600A A9 FF       30           LDA #$FF
600C 9D B8 04    31           STA PWDTH,X
600F 9D 38 06    32           STA ESCHR,X
6012 A9 00       33           LDA #0
6014 9D B8 05    34           STA MODE,X
6017 20 49 61    35           JSR DISPLAY
601A 8D          36           HEX 8D
601B 1D          37           HEX 1D       *16.5 Characters/inch
601C 1B 38       38           HEX 1B38     *8 Lines/inch
601E 1B 42       39           HEX 1B42     *105 Characters/line
6020 8D 00       40           HEX 8D00
6022             41  **********************
6022             42  * Begin main routine *
6022             43  **********************
6022 A9 00       44           LDA #0
6024 8D 42 61    45           STA COLNUM
6027 8D 43 61    46           STA COLNUM+1
602A AD 41 61    47  NXTLINE  LDA PAGE
602D C9 40       48           CMP #$40
602F F0 04       49           BEQ PAGE2
6031 A9 9F       50           LDA #159
6033 D0 02       51           BNE SETPAGE
6035 A9 BF       52  PAGE2    LDA #191
6037 8D 44 61    53  SETPAGE  STA ROWNUM
603A 8D 59 C0    54           STA HIGHON   *Turn on Graphics
603D A0 02       55  NXTROW   LDY #2
603F 20 D2 60    56  FORMGR   JSR SCRNCHK
6042 6E 48 61    57           ROR GRAPHIC
6045 CE 44 61    58           DEC ROWNUM
```

(continued)

```
6048 20 D2 60          JSR SCRNCHK
604B 6E 48 61          ROR GRAPHIC
604E EE 44 61          INC ROWNUM
6051 EE 42 61          INC COLNUM
6054 D0 03             BNE LOW
6056 EE 43 61          INC COLNUM+1
6059 88       LOW      DEY
605A 10 E3             BPL FORMGR
605C 38                SEC
605D 6E 48 61          ROR GRAPHIC
6060 38                SEC
6061 6E 48 61          ROR GRAPHIC
6064 AD 48 61          LDA GRAPHIC
6067 20 02 C1          JSR PRINTGR
606A AD 00 C0          LDA KBD
606D 10 07             BPL NOSTOP
606F 2C 10 C0          BIT STROBE
6072 C9 9B             CMP #$9B     *ESC pressed?
6074 F0 43             BEQ FINISH
6076 38       NOSTOP   SEC
6077 AD 42 61          LDA COLNUM
607A E9 03             SBC #3
607C 8D 42 61          STA COLNUM
607F AD 43 61          LDA COLNUM+1
6082 E9 00             SBC #0
6084 8D 43 61          STA COLNUM+1
6087 CE 44 61          DEC ROWNUM
608A CE 44 61          DEC ROWNUM
608D AD 44 61          LDA ROWNUM
6090 C9 FF             CMP #$FF
6092 D0 A9             BNE NXTROW
6094 8D 58 C0          STA HIGHOFF  *Turn off Graphics
6097 A9 8D             LDA #$8D
6099 20 02 C1          JSR PRINTGR
609C 18                CLC
609D AD 42 61          LDA COLNUM
60A0 69 03             ADC #3
60A2 8D 42 61          STA COLNUM
60A5 AD 43 61          LDA COLNUM+1
60A8 69 00             ADC #0
60AA 8D 43 61          STA COLNUM+1
60AD F0 07             BEQ ALLOK
60AF AD 42 61          LDA COLNUM
60B2 C9 17             CMP #23
60B4 F0 03             BEQ FINISH
60B6 4C 2A 60 ALLOK    JMP NXTLINE
60B9          **************************
60B9          * Reset printer defaults *
60B9          **************************
60B9 8D 58 C0 FINISH   STA HIGHOFF  *Turn off Graphics
60BC 20 49 61          JSR DISPLAY
60BF 8D                HEX 8D
60C0 1E                HEX 1E       *10 Characters/inch
60C1 1B 36             HEX 1B36     *6 Lines/inch
60C3 1B 41             HEX 1B41     *80 Characters/line
60C5 8D 8D 8D          HEX 8D8D8D
60C8 8D 8D 8D          HEX 8D8D8D
60CB 8D 8D 8D          HEX 8D8D8D
60CE 8D 8D 00          HEX 8D8D00
60D1 60                RTS
60D2          ***************************
60D2          * This is a high-res SCRN *
60D2          * function  The result is *
60D2          * returned in the carry   *
60D2          * flag  If the carry is   *
60D2          * clear, then no color    *
60D2          * is present; if the      *
60D2          * carry is set, then      *
60D2          * color is present        *
60D2          ***************************
60D2 98       SCRNCHK  TYA
60D3 48                PHA
60D4 A9 00
       LDA #0
60D6 8D 45 61          STA COLBYTE
60D9 AD 42 61          LDA COLNUM
60DC 8D 47 61          STA COLTEMP
60DF AD 43 61          LDA COLNUM+1
60E2 F0 0E             BEQ NOT>255
60E4 A9 24             LDA #36      *If X-coord is >255
60E6 8D 45 61          STA COLBYTE  * Add 36 to Byte#
60E9 18                CLC          * Add 4 to Low Byte
60EA AD 47 61          LDA COLTEMP  * of X-coord to
60ED 69 04             ADC #4       * compensate
60EF 8D 47 61          STA COLTEMP
60F2 AD 47 61 NOT>255  LDA COLTEMP
60F5 A2 00             LDX #0
60F7 38       DIVIDE7  SEC          *Calculate X/7
60F8 E9 07             SBC #7       *(Byte position)
```

```
60FA 90 03    147          BCC OVER
60FC E8       148          INX
60FD B0 F8    149          BCS DIVIDE7
60FF 18       150 OVER     CLC          *Put X MOD 7 in
6100 69 07    151          ADC #7       * 'COLBIT'
6102 8D 46 61 152          STA COLBIT   * (Bit position)
6105 18       153          CLC
6106 8A       154          TXA          *Calculate the
6107 6D 45 61 155          ADC COLBYTE  * Byte#
610A 8D 45 61 156          STA COLBYTE
610D          157 ****************************
610D          158 * Base address calculation *
610D          159 ****************************
610D AD 44 61 160          LDA ROWNUM   *Convert Row number
6110 48       161          PHA          * designated by RSTUVWXY
6111 29 C0    162          AND #$C0     * to  GBASL=VRSRS000
6113 85 26    163          STA GBASL    * and GBASH=PPPWXYTU
6115 4A       164          LSR          * where PPP=001 for Page 1
6116 4A       165          LSR          *          =010 for Page 2
6117 05 26    166          ORA GBASL
6119 85 26    167          STA GBASL    *GBASL becomes the pointer
611B 68       168          PLA          * to the left side of the
611C 85 27    169          STA GBASH    * current row
611E 0A       170          ASL
611F 0A       171          ASL
6120 0A       172          ASL
6121 26 27    173          ROL GBASH
6123 0A       174          ASL
6124 26 27    175          ROL GBASH
6126 0A       176          ASL
6127 66 26    177          ROR GBASL
6129 A5 27    178          LDA GBASH
612B 29 1F    179          AND #$1F
612D 0D 41 61 180          ORA PAGE
6130 85 27    181          STA GBASH
6132          182 ***************************
6132          183 * Put bit status in carry *
6132          184 ***************************
6132 AC 45 61 185          LDY COLBYTE
6135 B1 26    186          LDA (GBASL),Y
6137 AC 46 61 187          LDY COLBIT
613A 4A       188 GETBIT   LSR
613B 88       189          DEY
613C 10 FC    190          BPL GETBIT
613E 68       191          PLA
613F A8       192          TAY
6140 60       193          RTS
6141          194 *************************
6141          195 * PAGE = $20 for Page 1 *
6141          196 *      = $40 for Page 2 *
6141          197 *************************
6141 40       198 PAGE     HEX 40       *Page mask
6142          199 COLNUM   DFS 2
6144          200 ROWNUM   DFS 1
6145          201 COLBYTE  DFS 1
6146          202 COLBIT   DFS 1
6147          203 COLTEMP  DFS 1
6148          204 GRAPHIC  DFS 1
6149          205 **********************
6149          206 * Display subroutine *
6149          207 **********************
              208 MSG      EPZ $00
6149 A5 00    209 DISPLAY  LDA MSG      *Save 0-page
614B 8D 7C 61 210          STA ZPTEMP
614E A5 01    211          LDA MSG+1
6150 8D 7D 61 212          STA ZPTEMP+1
6153 68       213          PLA          *Pull the return address
6154 85 00    214          STA MSG      * from the stack (this address
6156 68       215          PLA          * is one less than the first
6157 85 01    216          STA MSG+1    * ASCII location)
6159 A0 00    217          LDY #0
615B E6 00    218 P1       INC MSG      *Move to next byte
615D D0 02    219          BNE P2       * of message
615F E6 01    220          INC MSG+1
6161 B1 00    221 P2       LDA (MSG),Y  *Get message byte
6163 F0 06    222          BEQ P3       *Done if zero
6165 20 02 C1 223          JSR PRINTGR
6168 4C 5B 61 224          JMP P1
616B A5 01    225 P3       LDA MSG+1    *Fix the return address
616D 48       226          PHA          * so that the opcode after
616E A5 00    227          LDA MSG      * the $00 terminator is
6170 48       228          PHA          * executed after return
6171 AD 7C 61 229          LDA ZPTEMP   *Restore 0-page
6174 85 00    230          STA MSG
6176 AD 7D 61 231          LDA ZPTEMP+1
6179 85 01    232          STA MSG+1
617B 60       233          RTS
617C          234 ZPTEMP   DFS 2
617E          235          END
```

MICRO™

Circle No. 45

Circle No. 46

A Universal Printer Pager for APPLE

by C. Anthony Ray

PAGER is an assembly-language program for the Apple II that automatically produces top and bottom margins on each page of printer output. It works with any printer, and can be switched on and off from both immediate and deferred execution modes.

UNIVERSAL PAGER
requires:
Apple with printer

Have you ever stopped your printer because it was printer on the perforations between pages? Do you find something distasteful about program listings and outputs that run off the bottom of one page and onto the top of the next? Have you wished someone would publish an easy fix that would work for *your* Apple and *your* printer? PAGER helps you solve these problems.

How to Use PAGER

In immediate execution mode follow these steps:

1. Type PR#n where n is the number of the slot containing your printer interface card.
2. Type BRUN PAGER.OBJ0.
3. PAGER is now connected but not active. You can do whatever you wish and PAGER will not interfere unless you type CTRL-A, change the DOS I/O hooks, or overwrite Apple's page three of memory.
4. Move your printer to the top of a new page and type CTRL-A. The printer immediately moves down five lines.
5. Do whatever you wish, within the limits mentioned in step three — perhaps LIST a program. PAGER now produces four-line margins at both the top and bottom of each page.
6. Type CTRL-A again. PAGER automatically moves to the top of the next page and stops producing margins.
7. Go to step three.

CTRL-A is a toggle. Pressing CTRL-A once enables the margin function, and pressing CTRL-A again disables the function. To disconnect PAGER and the printer type PR#0.

To use PAGER in deferred execution mode (from within a BASIC program) you need the following lines in your program:

```
10 PRINT CHR$(4)"PR# n"
15 PRINT CHR$(12) : REM TOP OF
   FORM CMD — EPSON
20 PRINT CHR$(4)"BRUN PAGER.OBJ0"
```

Line 15 must contain at least one of your printer's setup commands. To toggle the function from within the program use PRINT CHR$(1);. The semicolon prevents a carriage return from being sent to the printer, which upsets the size of the margin. Everything else works the same as in immediate mode.

Background

The BASIC operating system communicates with the outside world by transmitting and receiving strings of characters. It is the responsibility of each peripheral device to provide its own routine to transmit characters to, or receive characters from, the BASIC operating system. Two pointers, one for the character-input routine (KSW or Keyboard-input SWitch, $38-39) and one for the character-output routine (CSW or Character-output SWitch, $36-37), are used by the BASIC operating system to locate the peripheral-driver routines that are currently being used. Normally these contain the addresses of the monitor's keyboard-input routine and the screen-output routine.

The DOS (Disk Operating System) is the driving routine for the disk. To save you from having to switch back and forth between the screen routine and the disk routine every time you want to use the disk, DOS contains an intercept routine that recognizes the commands SAVE, LOAD, etc. If a string of characters is a DOS command it is executed, and if it is not a DOS command it is passed on to the screen routine.

DOS places itself between the BASIC operating system and the screen-output routine by copying the CSW pointer to another location and

substituting the address of its intercept routine. Similarly, DOS saves the KSW pointer and replaces it with the address of one of its own routines.

Typing PR#n causes DOS to use $Cn00 as the address of its character-output routine. $Cn00 is the first ROM (Read Only Memory) address on the peripheral interface card in slot #n. Similarly, typing IN#n causes DOS to use $Cn00 as the address of its character-input routine.

PAGER is an additional intercept routine that is placed between DOS and whatever the current input and output routines are. PAGER inserts itself by first finding a location in DOS that holds the addresses of the character-input and output routines. These are copied into PAGER. Now PAGER places its own intercept routine addresses into the CSW and KSW locations. The final trick is getting DOS to accept these addresses as its new character-input and output routine addresses and place its own intercept addresses in the CSW and KSW locations. Apple has provided an easy way to do this. DOS contains a subroutine that does exactly what I want and its entry point has been placed at location $03EA. PAGER jumps to this subroutine, completing its installation.

Sometimes $Cn00 is not the address of the character-output routine in the interface ROM. When this is the case, at least one character must be sent to the interface after the PR#n command and before connecting PAGER so that the interface can provide DOS with the correct address. This causes a problem only when PAGER is used in deferred execution mode. The best cure I've found is sending the printer setup commands to the printer before BRUNning PAGER.OBJ0.

When debugging a program using PAGER, hit RESET or type PR#0 before RUNning the program. Otherwise, starting with the second RUN, PAGER overwrites a still active copy of itself, causing the system to hang.

How PAGER Works

After you type PR#n, DOS stores the address of the printer-driver routine. PAGER locates this address as well as the address of the character-input routine. For both DOS 3.2 and 3.3 these addresses occupy four consecutive bytes, which are found by adding $0B96 to the address in CSW location (lines 34-40). The character-input and printer-output routine addresses are inserted in PAGER (lines 41-54) and PAGER's intercept routine addresses are installed in DOS (lines 55-63).

The input-intercept routine (lines 73-99) looks for CTRL-A's. If one is found, FLAG is incremented by one and the least significant bit (LSB) is checked. If FLAG's LSB is 1, PAGER produces the top margin by setting MCT to TMARGIN and falling into the LOOP that outputs MCT carriage returns. If FLAG's LSB is 0, PAGER moves to the top of the next page by adding the number of lines left unprinted on the page (LCT) to the bottom margin, BMARGIN, storing the results in MCT and jumping into the LOOP (lines 100-104). The input-intercept routine must return a valid character to its caller that does not interfere with whatever is to follow — LIST, DOS command, etc. A blank is ignored by all the command interpreters, so I've chosen to return a blank. The input-intercept routine allows PAGER to work in immediate execution mode.

The output-intercept routine (lines 113-124) starts by checking for a CTRL-A and, if one is found, jumps to the routines just discussed. This provides the deferred execution mode of operation. Next PAGER checks for a carriage return. If one is found, PAGER looks to see if FLAG's LSB is 1 by shifting it into the carry register and testing it with a conditional branch. If it is 1, the lines-left count (LCT) is decremented. If LCT is 0, all the lines to be printed on the current page have been printed and LOOP is invoked to produce MCT (which was previously set to TMARGIN + BMARGIN) carriage returns. If any of the above tests fail, the charcter is sent on to the printer-character-output routine.

Customizing PAGER

You may change the top and bottom margins and the number of lines printed by changing TMARGIN, BMARGIN, and PL, respectively. If your printer uses CTRL-A as a function-selecter code you may choose something else for the toggle by changing the value of CTRLA.

A few printers have their driver routines on disk instead of in ROM. If the driver routine occupies the same memory area as PAGER, PAGER may be assembled to load at another location.

You may contact the author at Cactus Computer Company, 39 Carriage Place, Urbana, IL 61801.

```
                3  **************************
                4  *                        *
                5  *   UNIVERSAL PAGER      *
                6  *   BY C. ANTHONY RAY    *
                7  *                        *
                8  **************************
                9          ORG $300
               10  *
               11  *
               12  * PROGRAM CONSTANTS
               13  *
               14  PL      EQU 58
               15  TMARGIN EQU 5
               16  BMARGIN EQU 5
               17  CR      EQU $8D
               18  GTRLA   EQU $81
               19  CSWL    EPZ $36
               20  CSWH    EPZ CSWL+$1
               21  KSWL    EPZ CSWH+$1
               22  KSWH    EPZ KSWL+$1
               23  *
               24  *
               25  *INITIALIZATION ROUTINE
               26  *
               27  *FUNCTION ACTIVE FLAG SET TO INACTIVE
               28  *CHARACTER INPUT AND OUTPUT ROUTINE ADDRESSES COMPUTED
               29  *PAGER INSERTED BETWEEN DOS AND CHAR INPUT AND
               30  *OUTPUT ROUTINES
               31  *
0300 A9 00     32          LDA #0
0302 8D 9D 03  33          STA FLAG    *INITIALIZE ACTIVITY FLAG
0305 18        34          CLC         *COMPUTE LOCATION OUTPUT ROUTINE
0306 A5 36     35          LDA CSWL    *ADDRESS LOW ORDER BYTE
0308 69 96     36          ADC #$96
030A 85 36     37          STA CSWL
030C A5 37     38          LDA CSWH    *COMPUTE LOCATION OUTPUT ROUTINE
030E 69 0B     39          ADC #$0B    *ADDRESS HIGH ORDER BYTE
0310 85 37     40          STA CSWH
0312 A0 00     41          LDY #0
0314 B1 36     42          LDA (CSWL),Y*INSERT ADDRESS OUTPUT ROUTINE
```

(continued)

```
0316 8D 62 03          STA LOC1+1  *INTO PROGRAM - LOW ORDER BYTE
0319 8D 9B 03          STA LOC2+1
031C C8                INY
031D B1 36             LDA (CSWL),Y*INSERT ADDRESS OUTPUT ROUTINE
031F 8D 63 03          STA LOC1+2  *INTO PROGRAM - LOW ORDER BYTE
0322 8D 9C 03          STA LOC2+2
0325 C8                INY
0326 B1 36             LDA (CSWL),Y*INSERT ADDRESS INPUT ROUTINE
0328 8D 45 03          STA INPUT+1 *INTO PROGRAM - LOW ORDER BYTE
032B C8                INY
032C B1 36             LDA (CSWL),Y*INSERT ADDRESS INPUT ROUTINE
032E 8D 46 03          STA INPUT+2 *INTO PROGRAM - HIGH ORDER BYTE
0331 A9 44             LDA #INPUT  *REPLACE INPUT HOOKS WITH ADDRESS
0333 85 38             STA KSWL    *OF PAGER INPUT ROUTINE
0335 A9 03             LDA /INPUT
0337 85 39             STA KSWH
0339 A9 80             LDA #OUTPUT *REPLACE OUTPUT HOOKS WITH ADDRESS
033B 85 36             STA CSWL    *OF PAGER OUTPUT ROUTINE
033D A9 03             LDA /OUTPUT
033F 85 37             STA CSWH
0341 4C EA 03          JMP $3EA    *RECONNECT DOS
0344           *
0344           *
0344           *INPUT FILTER ROUTINE
0344           *
0344           *INPUT MONITORED FOR CTRL-A.  EACH CTRL-A ADDS ONE TO
0344           *FLAG.  IF LSB OF FLAG IS ONE PAGER FUNCTION IS ACTIVE
0344           *OTHERWISE PAGER FUNCTION IS INACTIVE
0344           *
0344           *
0344 20 FF FF  INPUT   JSR $FFFF   *GET INPUT CHARACTER
0347 C9 81             CMP #CTRLA  *IS IT A CTRL-A?
0349 D0 28             BNE RETURN  *NO - RETURN
034B EE 9D 03  LAB1    INC FLAG    *YES - INCREMENT FLAG
034E A9 01             LDA #1
0350 2C 9D 03          BIT FLAG    *FLAG SET?
0353 F0 1F             BEQ CLPAGE  *NO - NEW PAGE
0355 A9 05             LDA #TMARGIN*YES - PRODUCE TOP MAR OUTPUT ROUTINE
0357 8D 9E 03          STA MCT     *SET MARGIN SIZE COUNT
035A           *
035A           *
035A           *LOOP BELOW IS USED TO SEND CARRIAGE RETURNS TO THE
035A           *OUTPUT ROUTINE TO PROVIDE VERTICAL MARGIN
035A           *AND PAGE FUNCTIONS.
035A           *
035A           *
035A A9 8D     LOOP    LDA #CR
035C CE 9E 03          DEC MCT        *SEND MCT CR'S TO PRINTER
035F F0 06             BEQ DONE
0361 20 FF FF  LOC1    JSR $FFFF      *GO CHAR OUTPUT ROUTINE
0364 4C 5A 03          JMP LOOP
0367 A9 0A     DONE    LDA #TMARGIN+BMARGIN
0369 8D 9E 03          STA MCT        *RESET MARGIN SIZE COUNT
036C A9 3A             LDA #PL
036E 8D 9F 03          STA LCT        *RESET PRINTED LINE COUNT
0371 A9 A0             LDA #$A0       *RETURN VALID INPUT CHAR
0373 60        RETURN  RTS
0374 18        CLPAGE  CLC            *COMPUTE NUMBER OF LINES
0375 AD 9F 03          LDA LCT        *LEFT ON PAGE
0378 69 05             ADC #BMARGIN
037A 8D 9E 03          STA MCT
037D 4C 5A 03          JMP LOOP       *MOVE TO TOP OF NEW PAGE
0380           *
0380           *
0380           *OUTPUT FILTER ROUTINE
0380           *OUTPUT IS MONITORED FOR CARRIAGE RETURNS.
0380           *WHEN PL CARRIAGE RETURNS ARE COUNTED PAGING
0380           *OCCURS.
0380           *
0380           *
0380 C9 81     OUTPUT  CMP #CTRLA     *CONTROL-A?
0382 F0 C7             BEQ LAB1       *YES - TOGGLE FLAG
0384 C9 8D             CMP #CR        *CARRIAGE RETURN?
0386 D0 12             BNE LOC2       *NO - GOTO CHAR OUTPUT ROUTINE
0388 4E 9D 03          LSR FLAG       *FLAG SET?
038B 90 0A             BCC LAB2       *NO - PAGER NOT ACTIVE
038D 2E 9D 03          ROL FLAG       *YES - RESTORE FLAG
0390 CE 9F 03          DEC LCT        *DECREMENT LINES-TO-PRINT COUNT
0393 D0 05             BNE LOC2       *LAST LINE? NO-GOTO OUTPUT ROUTINE
0395 F0 C3             BEQ LOOP       *YES - MOVE TO NEW PAGE
0397 2E 9D 03  LAB2    ROL FLAG       *RESTORE FLAG
039A 4C FF FF  LOC2    JMP $FFFF      *JUMP TO CHAR OUTPUT ROUTINE
039D           FLAG    DFS 1          *ACTIVITY FLAG
039E           MCT     DFS 1          *MARGIN SIZE COUNT
039F           LCT     DFS 1          *LINES-TO-PRINT COUNT
03A0                   END
```

MICRO

Second Pet as Soft Printer

by Hans Hoogstraat

Address a second PET as IEEE device 4 — especially useful for previewing formatted printouts.

```
;--------------------------------------------------------
            ;------------- SIMULATE A PRINTER AS DEVICE 4 -------------
            ;
            ;                      BY HANS HOOGSTRAAT
            ;----------------------------------------------------------
            ;
            ;                    ASSEMBLED FOR ANY ROM
            ;
            ;----------------------------------------------------------
            ;BASIC STARTUP
            ;---------------------------
            ;
                        .OS
                        .BA $400
            ;
            ;10 SYS 1050
            ;
0400-000B04             .BY 0 $B 4 $A 0
0403-0A00
0405-9E3130             .BY $9E $31 $30 $35 $30
0408-3530
040A-000000             .BY 0 0 0
            ;
                        .BA 1050
            ;
            ;---------------------------
            ;
            ;IEEE COMMUNICATIONS BITS.
            ;
            ;BITS 07 06 05 04 03 02 01 00
            ;.... -----------------------
            ;... 128 64 32 16 08 04 02 01
            ;
            ;E820  . . . . . . . . . DATA
            ;
            ;E821  X . . . . . . . . ATN
            ;E821  . . . . X . . . . NDAC
            ;
            ;E840  X . . . . . . . . DAV
            ;E840  . . . . . . . X . NRFD
            ;
            DEVICE      .DI 4
            ;
            ;---------------------------
            ;
041A-A900               LDA #$00
041C-8DED04             STA NCHAR
041F-8DEE04             STA LINES
            ;
            ;---------------------------
            ;WAIT FOR IEEE PRINTER SELECT.
            ;---------------------------
            ;
```

Simulate a Printer

For those who are fortunate enough to own two PET computers, this routine allows one PET to behave like a printer, addressable as device 4 from the other PET. One application is as a soft printer for debugging formatted printouts. It eliminates the time-consuming check-out of massive hard copy printouts.

I use the 80-column screen of my SuperPET (operating in the 6502 mode) to check the output from WordPro 3, operating on my 40-column PET. I considered buying WordPro 4, which has a similar preview feature, but the reduced text capacity (150 80-column lines *versus* 350 40-column lines) and high price changed my mind. I simply attach the PETs together using IEEE cables and load the printer simulator into the SuperPET. This gives me great flexibility and fast checkout capabilities of WordPro's output, without wasting a lot of paper. The routine is rather simple. It demonstrates how to program the PET IEEE bus for an addressed listening device. Secondary address information is ignored but could be picked up for sub-functions.

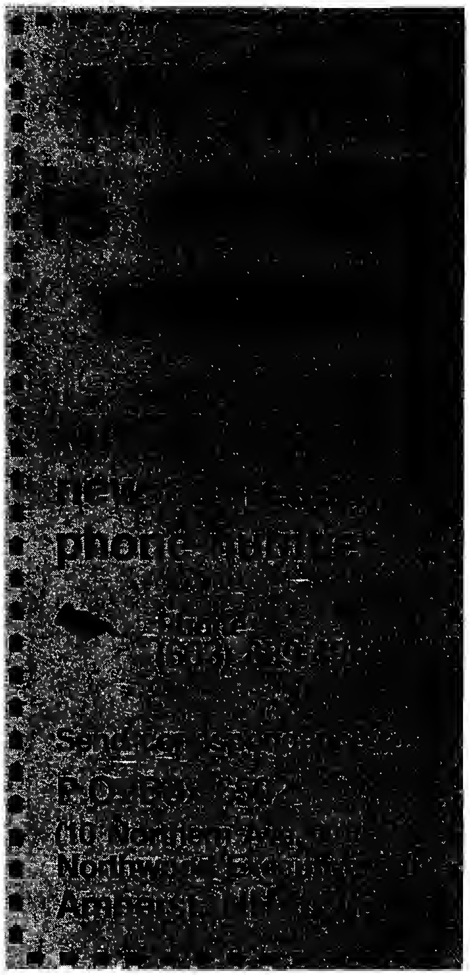
The program is set up to wait for any key reply after every 25 lines of displayed printout. If you want, the real printer can be active at the same time. This technique could be programmed interrupt driven for both PET computers. Such an implementation opens a number of exciting applications, including two-user interactive games and downloading of programs or data from one computer to the other for shared or interactive processing.

(Listing on next page)

You may contact Mr. Hoogstraat at Box 20, Site 7, SS 1, Calgary, Alberta T2M 4N3, Canada.

```
            0530;WAIT FOR ATTENTION HIGH.
            0540;
0422-78     0550PRINTER    SEI
0423-AD21E8 0560           LDA E821
0426-10FA   0570           BPL PRINTER
            0580;
            0590;SET NDAC HIGH.
            0600;
0428-AD21E8 0610           LDA E821
042B-0908   0620           ORA #$08
042D-8D21E8 0630           STA E821
            0640;
0430-20B804 0650           JSR GETIEEE
            0660;
            0670;CHECK IF PRINTER DEVICE.
            0680;
0433-C024   0690           CPY #DEVICE$20
0435-D0EB   0700           BNE PRINTER
            0710;
            0720;PRINTER SELECTED.
            0730;
0437-F003   0740           BEQ SHKBYTE
            0750;
            0760;------------------------------
            0770;GET A INPUT BYTE.
            0780;------------------------------
            0790;
0439-20B804 0800GETBYTE    JSR GETIEEE
            0810;
043C-20CF04 0820SHKBYTE    JSR SHKIEEE
            0830;
            0840;DATA INFO ON ATTENTION LOW.
            0850;
043F-8A     0860           TXA
0440-100D   0870           BPL CKDATA
            0880;
            0890;CHECK IF UNLISTEN.
            0900;
0442-C03F   0910           CPY #$3F
0444-F0DC   0920           BEQ PRINTER
            0930;
            0940;CHECK FOR CLOSE.
            0950;
0446-A8     0960           TAY
0447-29F0   0970           AND #$F0
0449-C9E0   0980           CMP #$E0
044B-F0D5   0990           BEQ PRINTER
            1000;
            1010;IGNORE SEC ADDRESSES ETC.
            1020;
044D-D0EA   1030           BNE GETBYTE
            1040;
            1050;VALID DATA BYTE.
            1060;
044F-C00D   1070CKDATA     CPY #$0D
0451-D02F   1080           BNE CKDATA2
            1090;
            1100;ON CAR.RET INGNORE IT ON CC  80
            1110;
0453-ADED04 1120           LDA NCHAR
0456-C950   1130           CMP #80
0458-D002   1140           BNE CKDATA1
            1150;
045A-A000   1160           LDY #$00
            1170;
045C-A900   1180CKDATA1    LDA #$00
045E-8DED04 1190           STA NCHAR
            1200;
            1210;ON CAR.RET INCREMENT LINE COUNT.
            1220;
0461-EEEE04 1230           INC LINES
0464-ADEE04 1240           LDA LINES
0467-C919   1250           CMP #25
0469-9042   1260           BCC PRDATA
            1270;
            1280;END OF SCREEN, RING BELL AND WAIT.
            1290;
046B-98     1300           TYA
046C-48     1310           PHA
046D-58     1320           CLI
046E-A907   1330           LDA #$07
0470-20D2FF 1340           JSR PRINT
            1350;
0473-20E4FF 1360WAIT       JSR GETCH
0476-F0FB   1370           BEQ WAIT
            1380;
0478-78     1390           SEI
0479-68     1400           PLA
047A-A8     1410           TAY
047B-A900   1420           LDA #$00
047D-8DEE04 1430           STA LINES
0480-F02B   1440           BEQ PRDATA
            1450;
            1460;CONVERT ATT. TO CLEAR SCREEN.
            1470;
0482-C007   1480CKDATA2    CPY #$07
0484-D00C   1490           BNE CKDATA3
            1500;
0486-A900   1510           LDA #$00
0488-8DED04 1520           STA NCHAR
048B-8DEE04 1530           STA LINES
            1540;
921-1

048E-A093   1550           LDY #$93
0490-D01B   1560           BNE PRDATA
            1570;
            1580;ACCEPT REVERSE ON AND OFF.
            1590;
0492-C012   1600CKDATA3    CPY #$12
0494-F017   1610           BEQ PRDATA
0496-C092   1620           CPY #$92
0498-F013   1630           BEQ PRDATA
            1640;
            1650;ACCEPT ASCII ONLY $20-$5F $A0-$DF
            1660;
049A-C020   1670           CPY #$20
049C-9098   1680           BCC GETBYTE
049E-C05F   1690           CPY #$5F
04A0-9008   1700           BCC ASCII
04A2-C0A0   1710           CPY #$A0
04A4-9093   1720           BCC GETBYTE
04A6-C0E0   1730           CPY #$E0
04A8-B08F   1740           BCS GETBYTE
            1750;
04AA-EEED04 1760ASCII      INC NCHAR
            1770;
            1780;PRINT DATA.
            1790;
04AD-98     1800PRDATA     TYA
04AE-F089   1810           BEQ GETBYTE
            1820;
04B0-58     1830           CLI
04B1-20D2FF 1840           JSR PRINT
04B4-78     1850           SEI
04B5-18     1860           CLC
04B6-9081   1870           BCC GETBYTE
            1880;
            1890;------------------------------
            1900;GET A BYTE FROM THE IEEE BUS.
            1910;------------------------------
            1920;
            1930;X  ATTENTION BYTE.
            1940;Y  IEEE DATA BYTE.
            1950;
            1960;SET NRFD HIGH.
            1970;
04B8-AD40E8 1980GETIEEE    LDA E840
04BB-0902   1990           ORA #$02
04BD-8D40E8 2000           STA E840
            2010;
            2020;WAIT FOR DAV LOW.
            2030;
04C0-AD40E8 2040WDAVL      LDA E840
04C3-30FB   2050           BMI WDAVL
            2060;
            2070;SAVE ATTENTION.
            2080;
04C5-AE21E8 2090           LDX E821
            2100;
            2110;GET BYTE AND COMPLEMENT IT.
            2120;
04C8-AD20E8 2130           LDA E820
04CB-49FF   2140           EOR #$FF
04CD-A8     2150           TAY
04CE-60     2160           RTS
            2170;
            2180;------------------------------
            2190;COMPLETE IEEE HANDSHAKE.
            2200;------------------------------
            2210;
```

Continued



```
              2220;SET NRFD LOW.
              2230;
04CF-AD40E8   2240SHKIEEE       LDA E840
04D2-29FD     2250              AND #$FF-02
04D4-8D40E8   2260              STA E840
              2270;
              2280;SET NDAC HIGH.
              2290;
04D7-AD21E8   2300              LDA E821
04DA-0908     2310              ORA #$08
04DC-8D21E8   2320              STA E821
              2330;
              2340;WAIT FOR DAV HIGH.
              2350;
04DF-AD40E8   2360WDAVH         LDA E840
04E2-10FB     2370              BPL WDAVH
              2380;
              2390;SET NDAC LOW.
              2400;
04E4-AD21E8   2410              LDA E821
04E7-29F7     2420              AND #$FF-08
04E9-8D21E8   2430              STA E821
04EC-60       2440              RTS
              2450;
              2460;-------------------------------
              2470;
              2480E820          .DI $E820
              2490E821          .DI $E821
              2500E840          .DI $E840
              2510;
              2520PRINT         .DE $FFD2
              2530GETCH         .DE $FFE4
              2540;
04ED-         2550NCHAR         .DS 1
04EE-         2560LINES         .DS 1
              2570              .EN

              LABEL FILE:

              DEVICE    0004      PRINTER   0422
              GETBYTE   0439      SHKBYTE   043C
              CKDATA    044F      CKDATA1   045C
              WAIT      0473      CKDATA2   0482
              CKDATA3   0492      ASCII     04AA
              PRDATA    04AD      GETIEEE   04B8
              WDAVL     04C0      SHKIEEE   04CF
              WDAVH     04DF      E820      E820
              E821      E821      E840      E840
              *PRINT    FFD2      *GETCH    FFE4
              NCHAR     04ED      LINES     04EE
```

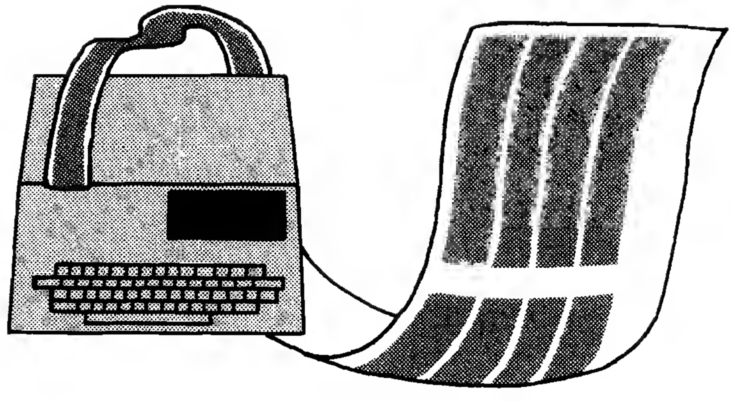
Hexadecimal dump Printer simulator.

```
--------------------------------------
0400 00 0B 04 0A 00 9E 31 30
0408 35 30 00 00 00 AA AA AA
0410 AA AA AA AA AA AA AA AA
0418 AA AA A9 00 8D ED 04 8D
0420 EE 04 78 AD 21 E8 10 FA
0428 AD 21 E8 09 08 8D 21 E8
0430 20 B8 04 C0 24 D0 EB F0
0438 03 20 B8 04 20 CF 04 8A
0440 10 0D C0 3F F0 DC A8 29
0448 F0 C9 E0 F0 D5 D0 EA C0
0450 0D D0 2F AD ED 04 C9 50
0458 D0 02 A0 00 A9 00 8D ED
0460 04 EE EE 04 AD EE 04 C9
0468 19 90 42 98 48 58 A9 07
0470 20 D2 FF 20 E4 FF F0 FB
0478 78 68 A8 A9 00 8D EE 04
0480 F0 2B C0 07 D0 0C A9 00
0488 8D ED 04 8D EE 04 A0 93
0490 D0 1B C0 12 F0 17 C0 92
0498 F0 13 C0 20 90 9B C0 5F
04A0 90 08 C0 A0 90 93 C0 E0
04A8 B0 8F EE ED 04 98 F0 89
04B0 58 20 D2 FF 78 18 90 81
04B8 AD 40 E8 09 02 8D 40 E8
04C0 AD 40 E8 30 FB AE 21 E8
04C8 AD 20 E8 49 FF A8 60 AD
04D0 40 E8 29 FD 8D 40 E8 AD
04D8 21 E8 09 08 8D 21 E8 AD
04E0 40 E8 10 FB AD 21 E8 29
04E8 F7 8D 21 E8 60
```

MICRO

MULTIC
A Multi-Column Printer for AIM

by Joel Swank



Print AIM listings the full width of your printer with this machine-language printer driver.

MULTIC
requires:
AIM with line printer

When I purchased my AIM 65, I was skeptical about the usefulness of the little onboard 20-column printer. I wondered what use it could have since I already had a full-sized printer. I soon realized how handy it is to have that small printer available since most AIM output is formatted to fit its 20-column line length. I even found it convenient to design some of my own programs to match the 20-column format. AIM's printer is fine for short- to medium-sized listings, but is inappropriate for long ones. I still send large listings to my full-size printer; but printing a 20-column listing on my printer, which can handle lines up to 132 columns long, wastes paper and time. To solve this problem I wrote MULTIC.

MULTIC inputs a single column of print from the active input device (AID) and reformats it into two to nine columns, side by side, on the page. This saves a lot of paper. I originally designed MULTIC to reformat 20-characer lines into pages 96 characters across with four columns; I decided MULTIC would be more useful if I made it as general as possible. MULTIC prompts for all information needed to describe the input and output. It must have enough RAM available to store one entire page of data. This RAM, called the page buffer, is allocated by the user and its size varies with the size of the output page.

Operation

MULTIC prompts for the following information:

FROM = The start of the page buffer in hexadecimal. The default is the end of MULTIC.

TO = The end of the page buffer in hexadecimal. The default is $FFFF.

INPUT LINE LENGTH = The maximum size of the input line in decimal. This is the width of one column on the output page. Input lines longer than this are split into two or more lines. Lines shorter than this are extended with blanks to keep the columns aligned. Minimum is 10, default is 20.

NUMBER OF COLUMNS = The number of columns of the input data to print across the output page. Default is 3.

SPACES BETWEEN COLUMNS = The number of blanks to insert between columns on the output page. Default is 5. The number of print columns needed to print a given format can be determined by the formula:

$$C * I + (C - 1) * S$$

where C is the number of columns, I is the input line length, and S is the number of spaces between columns. The result of this calculation must not exceed the column width of the printer that is to print the output.

LINES PER PAGE = The number of print lines per page. Maximum is 255, default is 60.

LINES BETWEEN PAGES = The number of lines to skip between pages. This number, plus the lines per page, should match the total number of lines the printer can print on one page of paper. If zero is entered for this prompt, MULTIC will send a form feed ($C) between pages instead of skipping lines to the top of the page. Default is 6.

IN = The standard AIM prompt to open the input device. The input file must be terminated with a control Z ($1A) so MULTIC can detect the end of the file.

OUT = The standard AIM prompt to open the output device. MULTIC will write the output to the AOD a page at a time. If the tape is the output device, it will be closed when MULTIC finishes.

Errors

Any errors detected during input of parameters are flagged with the standard AIM ERROR message and the - prompt re-issued. The MEMORY OVERFLOW error occurs when the page buffer fills before an entire page has been read in. The program must be restarted with more RAM allocated to the page buffer or a smaller page format.

Recently I used MULTIC to print a complete disassembly of the 8K AIM BASIC ROM, which amounts to about 4500 lines. This would take over 60 feet of paper on the AIM printer; I printed it five columns across the page and it took only 11 pages!

Modifications

Since MULTIC uses the AIM AID and AOD, the data to be printed must be stored on an auxiliary device. MULTIC is separated into subroutines so it can be modified to work directly from the user output port and to send output directly to the printer. The initialization would have to be executed as part of the open routine, and the subroutines REDBUF and REDCOL rewritten to execute as the user output routine. The output calls would have to be changed to call the printer driver routines directly.

(Listing on next page)

You can contact the author at 12550 SW Colony #3, Beaverton, OR 97005.

Listing 1: MULTIC Assembly Listing

```
 ;    MULTIC : MULTI COLUMN PRINT FORMATER FOR  AIM

 ;     FUNCTION : TO REFORMAT A TEXT FILE  FOR PRINTING
 ;       AS MULTI COLUMNS ON A PAGE. ALL INFO IS INPUT WITH
 ;      PROMPTS.

          .OPT GEN

     ;   AIM SUBROUTINES

     WHEREI =$E848          ;OPEN INPUT FILE
     WHEREO =$E871          ;OPEN OUTPUT FILE
     OUTALL =$E9BC          ;SEND CHARACTER TO AOD
     INALL  =$E993          ;GET CHAR FROM AID
     OUTPUT =$E97A          ;SEND CHAR TO DISPLAY
     CRLF   =$E9F0          ;SEND CR & LF TO AOD
     COMIN  =$E1A1          ;RETURN TO MONITOR
     CKER00 =$E394          ;DISPLAY 'ERROR'
     DU11   =$E50A          ;CLOSE TAPE FILE
     RDRUB  =$E95F          ;INPUT CHAR WITH DELETE
     RCHEK  =$E907          ;CHECK FOR INTERRUPT
     FROM   =$E7A3          ;DISPLAY 'FROM=' AND GET ADD
     TO     =$E7A7          ;DISPLAY 'TO=' AND GET ADD

     ;   ZERO PAGE

     START  =0              ;EIGHT BUFFER POINTERS
     LINLEN =$10            ;CHARACTERS PER COLUMN
     INTSPA =$11            ;SPACES BETWEEN COLUMNS
     COLUMN =$12            ;NUMBER OF COLUMNS
     LINES  =$13            ;LINES PER PAGE
     SKPLIN =$14            ;LINES TO SKIP AT END OF PAGE
     EOFLAG =$15            ;NON-ZERO FLAGS END OF FILE
     LCOUNT =$16            ;TEMP LINE COUNT
     CCOUNT =$17            ;TEMP CHAR COUNT
     DCOUNT =$18            ;DECIMAL CONVERSION TEMP
     DECTMP =$19            ;DECIMAL CONVERSION TEMP
     POINT  =$1A            ;BUFFER FILL POINTER
     BUFBEG =$1C            ;BEGINNING OF BUFFER
     BUFEND =$1E            ;END OF BUFFER
     SAVY   =$20            ;TEMP SAVE FOR Y
     SPLFLG =$21            ;SPLIT LINE FLAG

     ;   EQUATES

     LF     =10
     FF     =12
     CR     =13
     CTLZ   =$1A
     LINMAX =20             ;
     ;   DEFAULTS FOR PARMS
     LENDEF =20             ;WIDTH OF COLUMN
     INTDEF =5              ;SPACE BETWEEN COLUMNS
     COLDEF =3              ;NUMBER OF COLUMNS
     LINDEF =60             ;LINES PER PAGE
     SKPDEF =6              ;LINES BETWEEN PAGES

     ;   AIM RAM

     CKSUM  =$A41E
     CURAD  =$A41C

            *=$200

0200  20 F2 03  MULTIC JSR DEFALT     ;SET DEFAULTS
0203  20 4F 03         JSR GETPRM     ;INPUT INFO
0206  20 48 E8         JSR WHEREI     ;OPEN INPUT
0209  20 71 E8         JSR WHEREO     ;OPEN OUTPUT FILE
020C  A9 00            LDA #0         ;CLEAR SPLIT FLAG
020E  85 21            STA SPLFLG
0210  20 24 03  PAGLUP JSR ZEROPT     ;CLEAR POINTERS
0213  20 4B 02         JSR REDBUF     ;FILL PAGE BUFFER
0216            ;   PRINT THE BUFFER
0216  A5 13            LDA LINES      ;GET LINE COUNT
0218  85 16            STA LCOUNT
021A  A5 01     LINLUP LDA START+1    ;END OF FILE?
021C  F0 1C            BEQ ALLDUN     ;YES, QUIT
021E  20 D7 02         JSR PRTLIN     ;PRINT A LINE
0221  C6 16            DEC LCOUNT     ;COUNT IT
0223  D0 F5            BNE LINLUP     ;NEXT LINE
0225  A6 14            LDX SKPLIN     ;GET LINES TO SKIP
0227  F0 08            BEQ PAGUP      ;NONE, NEXT PAGE
0229  20 F0 E9  SKPLUP JSR CRLF       ;SKIP A LINE
022C  CA               DEX
022D  D0 FA            BNE SKPLUP
022F  F0 05            BEQ CKEND
```

Listing 1 *(Continued)*

```
0231  A9 0C     PAGUP  LDA #FF        ;SEND FORM FEED
0233  20 BC E9         JSR OUTALL
0236  A5 15     CKEND  LDA EOFLAG     ;END OF PAGE - ANY MORE DATA?
0238  F0 D6            BEQ PAGLUP     ;YES, CONTINUE

023A  A9 1A     ALLDUN LDA #CTLZ      ;SEND EOF CHAR
023C  20 BC E9         JSR OUTALL
023F  20 F0 E9         JSR CRLF
0242  20 F0 E9         JSR CRLF       ;AND 2 RETURNS
0245  20 0A E5         JSR DU11       ;CLOSE IF TAPE FILE
0248  4C A1 E1         JMP COMIN


024B            ;   END OF MAINLINE

024B            ;   SUBROUTINES FOLLOW

024B            ;   REDBUF  : READ PAGE BUFFER FULL WITH REQUESTED
024B            ;             NUMBER OF COLUMNS OF DATA

024B  A4 12     REDBUF LDY COLUMN     ;GET COLUMN COUNT
024D  A2 00            LDX #0         ;POINT TO FIRST BUFFER POINTER
024F  20 66 02  RDBLUP JSR REDCOL     ;READ A COLUMN
0252  A5 15            LDA EOFLAG     ;END?
0254  D0 0F            BNE RDBEND     ;YES, QUIT
0256  88               DEY            ;COUNT
0257  F0 0C            BEQ RDBEND     ;QUIT IF COUNT ZERO
0259  A5 1A            LDA POINT      ;COPY POINT TO
025B  95 02            STA START+2,X  ;NEXT START
025D  A5 1B            LDA POINT+1
025F  95 03            STA START+3,X
0261  E8               INX
0262  E8               INX            ;NEXT POINTER
0263  D0 EA            BNE RDBLUP     ;NEXT BUFFER
0265  60        RDBEND RTS


0266            ;   REDCOL  : READ A COLUMN INTO THE BUFFER

0266  A9 00     REDCOL LDA #0         ;CLEAR COUNTS
0268  85 17            STA CCOUNT
026A  85 16            STA LCOUNT


026C  20 93 E9  NEXTCH JSR INALL      ;GET A CHARACTER
026F  C9 0A            CMP #LF        ;LINE FEED?
0271  F0 F9            BEQ NEXTCH     ;YES, IGNORE IT
0273  C9 0C            CMP #FF        ;FORM FEED?
0275  F0 F5            BEQ NEXTCH     ;YES, IGNORE IT
0277  C9 0D            CMP #CR        ;END OF LINE?
0279  D0 0A            BNE NOCR       ;NOPE
027B  A5 21            LDA SPLFLG     ;FOLLOWING SPLIT LINE?
027D  F0 1B            BEQ EOL        ;NO, END OF LINE
027F  A9 00            LDA #0         ;YES, CLEAR FLAG
0281  85 21            STA SPLFLG
0283  F0 E7            BEQ NEXTCH     ;AND IGNORE IT
0285  20 AF 02  NOCR   JSR STOCH      ;SAVE IT
0288  C9 1A            CMP #CTLZ      ;END OF FILE?
028A  F0 20            BEQ EOF        ;YES
028C  A9 00            LDA #0         ;CLEAR SPLIT FLAG
028E  85 21            STA SPLFLG
0290  E6 17            INC CCOUNT     ;COUNT CHAR
0292  A5 17            LDA CCOUNT     ;GET MAX LENGTH
0294  C5 10            CMP LINLEN     ;AT MAX?
0296  90 D4            BCC NEXTCH     ;NO, GET ANOTHER
0298  85 21            STA SPLFLG     ;FLAG SPLIT LINE
029A  A9 0D     EOL    LDA #CR        ;YES, INSERT CR
029C  20 AF 02         JSR STOCH
029F  E6 16            INC LCOUNT     ;COUNT LINE
02A1  A9 00            LDA #0
02A3  85 17            STA CCOUNT     ;CLEAR CHAR COUNT
02A5  A5 16            LDA LCOUNT     ;GET LINE COUNT
02A7  C5 13            CMP LINES      ;AT LINE LIMIT?
02A9  D0 C1            BNE NEXTCH     ;NO, GET ANOTHER
02AB  60               RTS

02AC  85 15     EOF    STA EOFLAG     ;SIGNAL END
02AE  60               RTS


02AF            ;   STOCH ; STORE ONE CHARACTER IN BUFFER

02AF  84 20     STOCH  STY SAVY       ;SAVE Y
02B1  A0 00            LDY #0
02B3  91 1A            STA (POINT),Y  ;STORE CHAR
02B5  D1 1A            CMP (POINT),Y  ;DID IT TAKE?
02B7  D0 16            BNE MEMERR     ;NOPE
```

(continued)

Listing 1 *(Continued)*

```
02B9  A4 20            LDY SAVY        ;RESTORE Y
02BB  E6 1A            INC POINT       ;YES, BUMP POINTER
02BD  D0 02            BNE NOCY1
02BF  E6 1B            INC POINT+1
02C1  48        NOCY1  PHA
02C2  38               SEC
02C3  A5 1E            LDA BUFEND      ;CHECK FOR END OF BUFFER
02C5  E5 1A            SBC POINT
02C7  A5 1F            LDA BUFEND+1
02C9  E5 1B            SBC POINT+1
02CB  90 02            BCC MEMERR
02CD  68               PLA
02CE  60               RTS

02CF  A0 63     MEMERR LDY #MEMMSG-LITS
02D1  20 18 04         JSR KEPX        ;DISPLAY MEMORY MESSAGE
02D4  4C A1 E1         JMP COMIN       ;RETURN TO AIM


02D7            PRTLIN : PRINT A LINE FROM PAGE BUFFER

02D7  20 07 E9  PRTLIN JSR RCHEK       ;CHECK FOR INTERRUPT
02DA  A9 00            LDA #0          ;CLEAR COUNT
02DC  85 17            STA CCOUNT
02DE  A4 12            LDY COLUMN      ;GET COLUMN COUNT
02E0  A2 00            LDX #0          ;CLEAR POINTER INDEX
02E2  F0 16            BEQ PRCHR

02E4  A5 17     PRTLUP LDA CCOUNT      ;SEND SPACES TO NEXT COLUMN
02E6  C5 10     SPACLP CMP LINLEN      ;AT LINE LENTH?
02E8  F0 09            BEQ ISPACE      ;YES, QUIT
02EA  20 3A 03         JSR BLANK       ;SEND A BLANK
02ED  E6 17            INC CCOUNT      ;COUNT
02EF  A5 17            LDA CCOUNT      ;GET COUNT
02F1  D0 F3            BNE SPACLP      ;REPEAT

02F3  20 3F 03  ISPACE JSR SPCOUT      ;SEND INTER-COLUMN GAP
02F6  A9 00            LDA #0          ;CLEAR COUNT
02F8  85 17            STA CCOUNT
02FA  A1 00     PRCHR  LDA (START,X)   ;GET A CHAR
02FC  F6 00            INC START,X     ;BUMP POINTER
02FE  D0 02            BNE NOCY2
0300  F6 01            INC START+1,X
0302  C9 0D     NOCY2  CMP #CR         ;END OF LINE?
0304  F0 0B            BEQ COLEND      ;YES
0306  C9 1A            CMP #CTLZ       ;END OF PAGE?
0308  F0 12            BEQ LINEND      ;YES
030A  20 BC E9         JSR OUTALL      ;SEND IT
030D  E6 17            INC CCOUNT      ;COUNT CHAR
030F  D0 E9            BNE PRCHR       ;NEXT

0311  88        COLEND DEY             ;COUNT COLUMN
0312  F0 0C            BEQ CROUT       ;QUIT ON LAST
0314  E8               INX
0315  E8               INX             ;NEXT POINTER
0316  B5 01            LDA START+1,X   ;ANY DATA?
0318  D0 CA            BNE PRTLUP      ;YES
031A  F0 04            BEQ CROUT       ;NO

031C  A9 00     LINEND LDA #0          ;MARK THIS COLUMN FINISHED
031E  95 01            STA START+1,X
0320  20 F0 E9  CROUT  JSR CRLF        ;SEND CR & LF
0323  60               RTS


0324            ZEROPT : INIT POINTERS FOR NEXT PAGE

0324  A2 0F     ZEROPT LDX #15
0326  A9 00            LDA #0
0328  95 00     ZPLUP  STA START,X     ;CLEAR POINTER TABLE
032A  CA               DEX
032B  10 FB            BPL ZPLUP
032D  A5 1C            LDA BUFBEG      ;INIT FIRST POINTER
032F  85 00            STA START
0331  85 1A            STA POINT
0333  A5 1D            LDA BUFBEG+1
0335  85 01            STA START+1
0337  85 1B            STA POINT+1
0339  60               RTS

033A            BLANK : SEND SPACE TO AOD

033A  A9 20     BLANK  LDA #' '
033C  4C BC E9         JMP OUTALL

0218  033F             SPCOUT : SEND SPACES TO NEXT COLUMN
```

Listing 1 *(Continued)*

```
033F  A5 11     SPCOUT LDA INTSPA      ;GET SPACE COUNT
0341  F0 0B            BEQ SPCDWN      ;NONE, SKIPIT
0343  48               PHA             ;SAVE
0344  20 3A 03  SPCLUP JSR BLANK       ;SEND A BLANK
0347  C6 11            DEC INTSPA      ;COUNT
0349  D0 F9            BNE SPCLUP      ;REPEAT TILL ZERO
034B  68               PLA             ;RESTORE SPACE COUNT
034C  85 11            STA INTSPA
034E  60        SPCDWN RTS


034F            GETPRM : PROMPT FOR AND INPUT PARAMETERS

034F  20 A3 E7  GETPRM JSR FROM        ;GET BUFFER START
0352  B0 FB            BCS GETPRM
0354  AD 1E A4         LDA CKSUM       ;ANY ENTERED?
0357  D0 0A            BNE GTEND       ;NO, USE DEFAULT
0359  AD 1C A4         LDA CURAD       ;MOVE ADDRESS TO
035C  85 1C            STA BUFBEG      ;BEGIN POINTER
035E  AD 1D A4         LDA CURAD+1
0361  85 1D            STA BUFBEG+1

0363  20 A7 E7  GTEND  JSR TO          ;GET END ADDRESS
0366  B0 FB            BCS GTEND
0368  AD 1E A4         LDA CKSUM       ;ANY SPECIFIED?
036B  D0 0A            BNE GTLL        ;NO, USE DEFAULT
036D  AD 1C A4         LDA CURAD       ;YES, MOVE IT TO
0370  85 1E            STA BUFEND      ;END POINTER
0372  AD 1D A4         LDA CURAD+1
0375  85 1F            STA BUFEND+1

0377  20 F0 E9  GTLL   JSR CRLF        ;NEW LINE
037A  A0 00     RELL   LDY #LLMSG-LITS
037C  20 18 04         JSR KEPX        ;REQUEST LINE LENGTH
037F  20 24 04         JSR INDEC       ;INPUT A DECIMAL NUMBER
0382  B0 F6            BCS RELL        ;RETRY IF ERROR
0384  AE 1E A4         LDX CKSUM       ;ANY?
0387  D0 0C            BNE GTCOL       ;NO
0389  C9 0A            CMP #10         ;MIMIMUM OF 10
038B  B0 06            BCS STLEN
038D  20 94 E3         JSR CKER00      ;ERROR MSG
0390  4C 7A 03         JMP RELL        ;RETRY
0393  85 10     STLEN  STA LINLEN

0395  A0 2B     GTCOL  LDY #COLMSG-LITS
0397  20 18 04         JSR KEPX        ;REQUEST # OF COLUMNS
039A  20 24 04         JSR INDEC       ;GET REPLY
039D  B0 F6            BCS GTCOL       ;RETRY ON ERROR
039F  AE 1E A4         LDX CKSUM       ;ANY?
03A2  D0 10            BNE GTSP        ;NO
03A4  C9 01            CMP #1          ;MINIMUM OF 1
03A6  90 04            BCC COLERR
03A8  C9 09            CMP #9          ;MAX OF 8
03AA  90 06            BCC STCOL
03AC  20 94 E3  COLERR JSR CKER00      ;ERROR MSG
03AF  4C 95 03         JMP GTCOL
03B2  85 12     STCOL  STA COLUMN      ;SAVE IT

03B4  A0 13     GTSP   LDY #SPMSG-LITS
03B6  20 18 04         JSR KEPX        ;REQUEST SPACE BETWEEN COLUMNS
03B9  20 24 04         JSR INDEC       ;GET REPLY
03BC  B0 F6            BCS GTSP        ;RETRY ERROR

03BE  AE 1E A4         LDX CKSUM       ;ANY?
03C1  D0 02            BNE GTLIN       ;NO
03C3  85 11            STA INTSPA      ;SAVE IT

03C5  A0 3E     GTLIN  LDY #LINMSG-LITS
03C7  20 18 04         JSR KEPX        ;REQUEST LINES PER PAGE
03CA  20 24 04         JSR INDEC       ;GET REPLY
03CD  B0 F6            BCS GTLIN
03CF  AE 1E A4         LDX CKSUM       ;ANY?
03D2  D0 0C            BNE GTPER       ;NO
03D4  C9 02            CMP #2          ;MINIMUM OF 2
03D6  B0 06            BCS STPER
03D8  20 94 E3         JSR CKER00      ;ERROR MSG
03DB  4C C5 03         JMP GTLIN       ;RETRY
03DE  85 13     STPER  STA LINES       ;SAVE IT

03E0  A0 4E     GTPER  LDY #PERMSG-LITS
03E2  20 18 04         JSR KEPX        ;REQUEST LINES BETWEEN PAGES
03E5  20 24 04         JSR INDEC       ;GET REPLY
03E8  B0 F6            BCS GTPER       ;RETRY ON ERROR
03EA  AE 1E A4         LDX CKSUM       ;ANY?
03ED  D0 02            BNE GTRT        ;NO
03EF  85 14            STA SKPLIN      ;SAVE IT
03F1  60        GTRT   RTS
```

(continued)

Listing 1 *(Continued)*

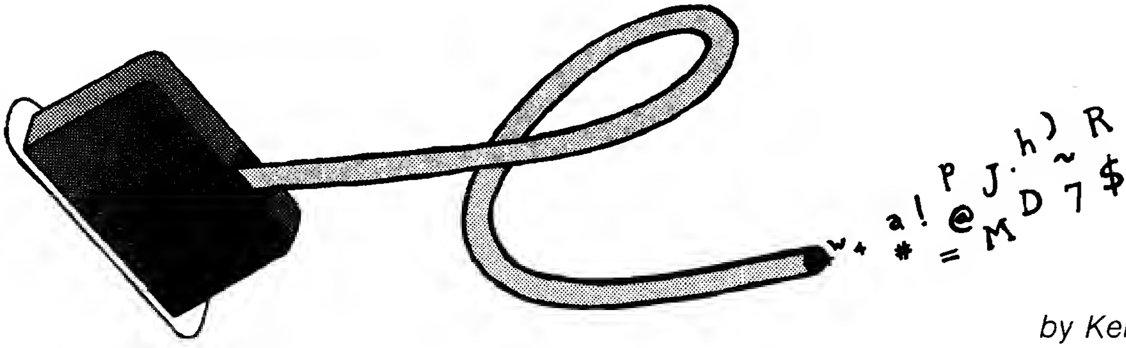
```
03F2              ;   DEFALT : INIT DATA AREAS WITH DEFAULTS

03F2  A9 14       DEFALT LDA #LENDEF
03F4  85 10              STA LINLEN
03F6  A9 05              LDA #INTDEF
03F8  85 11              STA INTSPA
03FA  A9 03              LDA #COLDEF
03FC  85 12              STA COLUMN
03FE  A9 3C              LDA #LINDEF
0400  85 13              STA LINES
0402  A9 06              LDA #SKPDEF
0404  85 14              STA SKPLIN
0406  A9 F9              LDA #<PAGBUF     ;PAGBUF IS DEFAULT FOR
0408  85 1C              STA BUFBEG       ;BUFFER BEGIN
040A  A9 04              LDA #>PAGBUF
040C  85 1D              STA BUFBEG+1
040E  A2 00              LDX #0
0410  86 15              STX EOFLAG       ;CLEAR EOF FLAG
0412  CA                 DEX
0413  86 1E              STX BUFEND       ;$FFFF IS END DEFAULT
0415  86 1F              STX BUFEND+1
0417  60                 RTS


0418              ;   KEPX : DISPLAY MESSAGE FROM TABLE

0418  B9 86 04    KEPX   LDA LITS,Y       ;GET A CHARACTER
041B  F0 06              BEQ KEPOUT       ;QUIT ON ZERO
041D  20 7A E9           JSR OUTPUT
0420  C8                 INY
0421  D0 F5              BNE KEPX
0423  60          KEPOUT RTS



0424              ;   INDEC : INPUT DECIMAL NUMBER FROM KEYBOARD

0424  20 70 04    INDEC  JSR INLIN        ;INPUT A LINE
0427  A9 00              LDA #0           ;START WITH ZERO
0429  85 18              STA DCOUNT
042B  8D 1E A4           STA CKSUM        ;ASSUME NO DEFAULT
042E  A2 00              LDX #0           ;CLEAR BUFFER INDEX
0430  BD F9 04    DECLUP LDA DECBUF,X     ;GET A CHARACTER

0433  C9 0D              CMP #CR          ;EOL?
0435  F0 2B              BEQ DECDON       ;YES
0437  C9 20              CMP #' '         ;END OF NUMBER?
0439  F0 27              BEQ DECDON       ;YES
043B  C9 30              CMP #'0'         ;VALID DECIMAL DIGIT?
043D  90 1B              BCC BADNUM       ;NO
043F  C9 3A              CMP #$3A
0441  B0 17              BCS BADNUM       ;NO
0443  29 0F              AND #$0F         ;CLEAR HIGH NYBBLE
0445  85 19              STA DECTMP       ;SAVE
0447  A5 18              LDA DCOUNT       ;MULTIPLY COUNT BY 10
0449  0A                 ASL A
044A  0A                 ASL A            ;TIMES 8
044B  0A                 ASL A
044C  18                 CLC
044D  65 18              ADC DCOUNT       ;AND ADD TWICE FOR 10
044F  18                 CLC
0450  65 18              ADC DCOUNT
0452  18                 CLC
0453  65 19              ADC DECTMP       ;ADD NEW DIGIT
0455  85 18              STA DCOUNT       ;SAVE
0457  E8                 INX
0458  D0 D6              BNE DECLUP       ;NEXT DIGIT

045A  20 94 E3    BADNUM JSR CKER00       ;DISPLAY 'ERROR'
045D  20 F0 E9           JSR CRLF
0460  38                 SEC

0461  60                 RTS
0462  8A          DECDON TXA              ;ANY ENDTERED?
0463  D0 04              BNE DECDUN       ;YES
0465  E8                 INX
0466  8E 1E A4           STX CKSUM        ;NO, SET FLAG
0469  20 F0 E9    DECDUN JSR CRLF         ;END LINE
046C  A5 18              LDA DCOUNT       ;GET COUNT
046E  18                 CLC
046F  60                 RTS

0470              ;   INLIN : INPUT A LINE OF CHARACTERS FROM KBD
0470              ;           AND STORE IN DECIMAL BUFFER

0470  A0 00       INLIN  LDY #0           ;CLEAR BUFFER INDEX
```

Listing 1 *(Continued)*

```
0472  20 5F E9    INLUP  JSR RDRUB        ;READ CHAR WITH DELETE
0475  99 F9 04           STA DECBUF,Y     ;SAVE CHARACTER
0478  C9 20              CMP #' '         ;SPACE OR CR ENDS LINE
047A  F0 09              BEQ ENDLIN
047C  C9 0D              CMP #CR
047E  F0 05              BEQ ENDLIN
0480  C8                 INY              ;BUMP INDEX
0481  C0 15              CPY #LINMAX+1    ;MAXIMUM CHARS?
0483  90 ED              BCC INLUP        ;NO, GET ANOTHER
0485  60          ENDLIN RTS


0486              ;      LITERAL TABLE

0486              LITS   =*


0486  49 4E       LLMSG  .BYTE 'INPUT LINE LENGTH=',0
0488  50 55
048A  54 20
048C  4C 49
048E  4E 45
0490  20 4C
0492  45 4E
0494  47 54
0496  48 3D
0498  00
0499  53 50       SPMSG  .BYTE 'SPACES BETWEEN COLUMNS=',0
049B  41 43
049D  45 53
049F  20 42
04A1  45 54
04A3  57 45
04A5  45 4E
04A7  20 43
04A9  4F 4C
04AB  55 3D
04AD  9B 94 9B
04B0  00
04B1  4E 55       COLMSG .BYTE 'NUMBER OF COLUMNS=',0


04B3  4D 42
04B5  45 52
04B7  20 4F
04B9  46 20
04BB  43 4F
04BD  4C 55
04BF  4D 4E
04C1  53 3D
04C3  00
04C4  4C 49       LINMSG .BYTE 'LINES PER PAGE=',0
04C6  4E 45
04C8  53 20
04CA  50 45
04CC  52 20
04CE  50 41
04D0  47 45 3D
04D3  00
04D4  4C 49       PERMSG .BYTE 'LINES BETWEEN PAGES=',0
04D6  4E 45
04D8  53 20
04DA  42 45
04DC  54 57
04DE  45 45
04E0  4E 20
04E2  50 41
04E4  47 45
04E6  53 3D
04E8  00
04E9  4D 45       MEMMSG .BYTE 'MEMORY OVERFLOW',0

04EB  4D 4F
04ED  52 59
04EF  20 4F
04F1  56 45
04F3  52 46
04F5  4C 4F 57
04F8  00


04F9              PAGBUF =*
04F9              DECBUF =*

04F9                     .END
04F9               ERRORS= 0000
```

MICRO

Superboard II / Quick Printer II



by Ken Benson

SB II/QP II
requires:
OSI Superboard II
Radio Shack Quick Printer II

This article describes an inexpensive way to interface an OSI Superboard II to a Radio Shack Quick Printer II.

After looking at a number of printers, I decided on the Radio Shack Quick Printer II, principally because of the reasonable price, but also because it was a complete package. A study of the schematic indicated that it should be possible to use the TRS-80 bus connector as a way into the Quick Printer II. A cable was included with the printer that could be used for this purpose, thus saving a few dollars.

Connected to the printer bus are 8 data lines, 16 address lines, an RD, and a WR line. The address lines decode to $37E8, which is the memory address for printer I/O in the TRS-80 Model I.

The Superboard II has a 40-pin expansion connector, which I have traced out; it seems that OSI forgot to put it on the schematic. The expansion connector includes 8 data lines, 16 address lines, and an R/W* line (the * denotes active low). I wondered if this would be enough to interface to the printer? If so, what means of addressing could be utilized? Would it be necessary to write all new software for the printer?

I thought it would be easy if I could use my Superboard II cassette routine that outputs to $F001, so I studied the address decoder in the Quick Printer II to see how difficult it would be to change the address. It looked like just switching a few leads would do it; in fact, I decided to try switching leads in the interconnect cable.

I removed one TRS-80 bus connector from the interconnect cable that had come with the Quick Printer II,

table 1

QP II Pin #	QP II Function	Goes To	SB II Function	SB II Pin #
4	A10	<——>	A15	27
5	A12	<——>	A14	26
6	A13	<——>	A13	25
7	A15*	<——>	GND	38
8	GND	<——>	GND	37
9	A11*	<——>	GND	10
10	A14*	<——>	GND	19
11	A8	<——>	A12	24
13	WR*	<——>	R/W*	32
17	A9	<——>	A0	14
18	D4	<——>	D4	36
20	D7	<——>	D7	33
22	D1	<——>	D1	5
24	D6	<——>	D6	34
25	A0*	<——>	A11*	23
26	D3	<——>	D3	7
27	A1*	<——>	A10*	22
28	D5	<——>	D5	35
29	GND	<——>	GND	29
30	D0	<——>	D0	4
31	A4*	<——>	A9*	21
32	D2	<——>	D2	6
37	GND	<——>	GND	28
40	A2*	<——>	A8*	20

Check the connections carefully because the [illegible] cable may not be numbered correctly. Note that pin number [illegible] on the QP II is on the [illegible] when facing the rear of the printer. Also U6 and U7 [illegible] populated.

and then attached a 40-pin DIP plug on the end, as shown in table 1. The RD line is not utilized, since it is not used in the TRS-80 alternate connection with the Expansion Interface. I thought operation would still be possible with this line deleted, and this proved to be the case. Crossing the lines fools the printer address-decoder logic into thinking it is addressed at $37E8 by the cassette output routine of Superboard II to $F001.

Unfortunately, a few minor problems occur. Because there is no handshake, the computer has to wait for the printer, or data will be lost. This means you must list or print in the "SAVE" mode, and you must set the terminal width to 31 using the command POKE 15,31. The "SAVE" mode then forces a carriage return and 10 nulls just as the printer is about to CR/LF automatically. This enables the printer to keep pace with the computer. Don't forget to reset line length to 72 before a cassette save or you will lose information upon reloading (the reloading program will have data after a carriage return and no line number for it).

Another problem is that you get a "?" for the first character of each line. I am not sure why this happens, but it could be because the "PRINT" token in SB II is "?". This is no problem for me, since I am interested only in program or disassembler listings. However, it does become difficult to print plain text or to use the large-character feature of the QP II (large characters are 15 per line, small are 32 per line), but it can be done. The trick is to write strings and POKE to the output. You can obtain the large letters in this manner. The specific procedure is as follows:

```
POKE 61441,13 (CR)
FOR W=0 TO 500 : NEXT (wait for
  printer to get ready)
POKE 61441,15 (shift to large format)
```

The desired message could then be printed in the following manner:

```
A$ = "TEXT TO BE PRINTED"
L = LEN(A$)
FOR N = 1 TO L
POKE MID$(A$,N,1)
NEXT N
```

I get two other bugs that may not bother other systems. I am using a modulator and a TV, so I get some interference on the screen. The other is that noise from the printer sometimes stops my program when I'm printing. Placement of the unit and cable affect this and I suspect it could be a grounding problem.

In summary, I am pleased with this configuration. It works well for program lists and for my disassembler. The silver paper is somewhat difficult to read because of the glare, but it copies nicely. Admittedly, this arrangement is not the most elegant. For someone on a tight budget, however, it might prove to be the perfect solution. Since hooking up this system I've seen a modification to double the baud rate of SB II from 300 to 600. With that modification installed the serial port might be used, although POKE might not work then. I've written a short routine that lets me print out what is being input to SB II, so I don't plan to make that change now; I may try it eventually for the faster loading.

K.L. Benson works for Hughes Aircraft Company as a microwave standards engineer. He may be contacted at 745 Venice Way, Apt. 1A, Inglewood, CA 90302.

MICRO™

Interface Clinic

by Ralph Tenny

The first two sessions in this series began laying the groundwork for interfacing experiments. At this point some readers need more information but others may be bored. So, to keep everyone interested, I will begin some simple experiments. A text on assembly language was recommended in last month's column, and I recommend the sources listed below as additional study material.

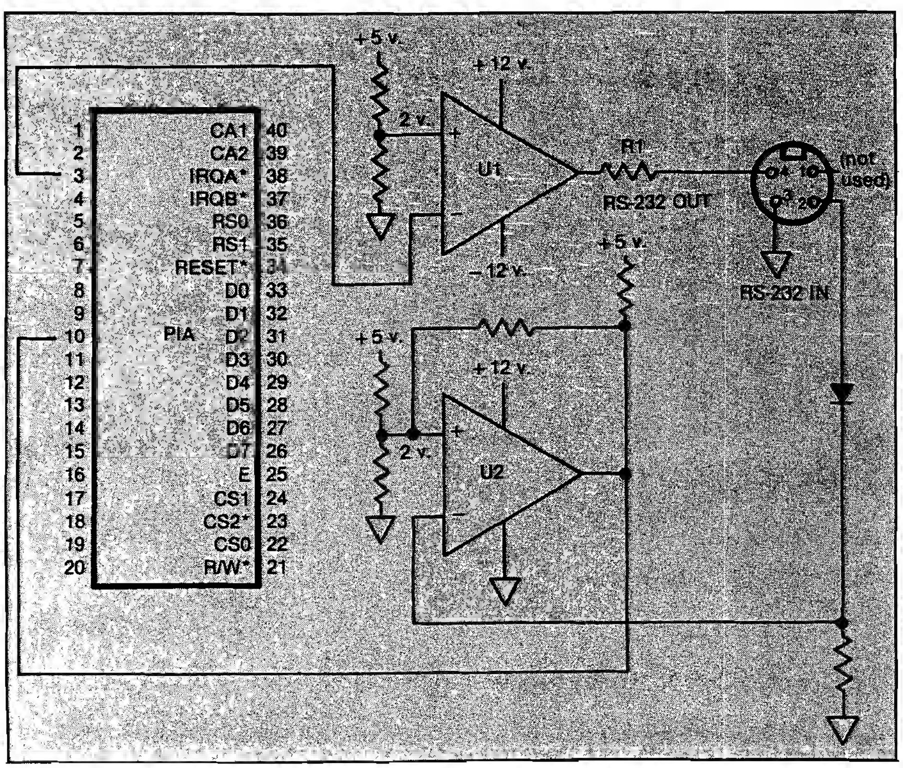
MICRO Cookbook, Don Lancaster, Howard W. Sams & Co., Inc. #21828
CMOS COOKBOOK, Don Lancaster, Howard W. Sams & Co., Inc. #21398
NCR Basic Electronics Course, Howard W. Sams & Co., Inc. #21549
Guide to CMOS Basics, Circuits & Experiments, Howard M. Berlin, Howard W. Sams & Co., Inc. #21654

Note that the references cited cover background material and do not deal with a specific computer. The NCR Basic Electronics Course is entry-level material and offers a broader range of useful topics than the other books. I recommend that CoCo owners obtain the Service Manual for the Color Computer (Radio Shack catalog #26-3001/3002), and that owners of other computers obtain similar hardware manuals for their own computer. I will mention various precautions regarding CoCo interfacing; these will be valid for other computers according to their particular design architecture.

As I move into the area of hardware experiments, you will need certain tools and materials to duplicate and expand upon the ideas presented. The only construction tools that you will need are those included in small kit-building sets available from Radio Shack and Heathkit. Accessories such as interface cables and similar hardware will be specified as the need arises. In general, experiments and interfaces will be designed around parts available from Radio Shack. Those who know how are encouraged to substitute whatever source is convenient; to this end both generic part numbers and Radio Shack catalog numbers will be used to specify parts used in the designs. If possible, each experimenter should obtain a small 5-volt supply to power these experiments; Radio Shack #277-125, a low-cost kit, will suffice. Often, a 6-volt lantern battery can be used for the experiments, including this month's circuits.

The Color Computer has only a few lines devoted to input and output (I/O). While you will eventually learn how to expand CoCo's I/O capability, first learn to use the existing capability. Our first experiment will use the serial port (printer port) for single-line output and input as a familiarization exercise. Figure 1 shows the output and input circuits used in the serial port. These circuits bridge the gap between the PIA used for I/O in the Color Computer and the outside world. That is, the PIA signal levels must be changed from digital logic signals to RS-232 signal levels.

Figure 1. **Serial I/O on the Color Computer is done with two amplifiers that change digital logic signal levels from the PIA to RS-232 levels. See text for more details.**



RS-232-level signals are used to interconnect almost all types of computer peripherals to the computers. RS-232 signals are defined by the voltage levels shown in figure 2; note that *valid* signals will be greater than +3 volts for a logic zero or less than −3 volts for logic one. The deadband between +3 volts and −3 volts is used for noise immunity on long signal paths that may have external interference.

Digital logic signals are typically defined as being less than 0.8 volts for logic 0 and greather than 2.2 volts (typically 3.5 volts) for logic 1 signals.

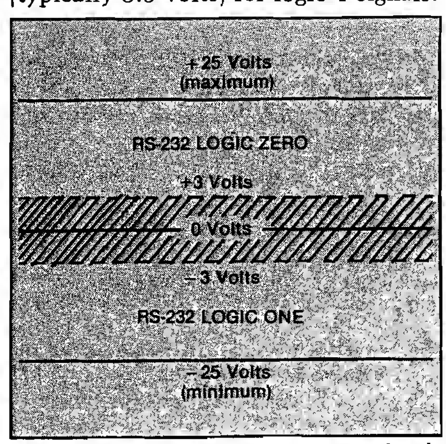


Figure 2. **Diagram showing voltage levels of RS-232 signal channels. Signal deadband between +3 volts and −3 volts gives noise immunity for long cable runs in noisy environments.**

U1 and U2 are special ICs that do the voltage translation from unipolar logic levels to bipolar RS-232 signals. U1's output can swing almost 12 volts plus and minus, with R1 limiting current to the external load. U2's output line is powered from the +5-volt bus, so it cannot swing higher than 5 volts; this avoids overdriving the PIA input line. Note that both U1 and U2 have dual inputs; one input is held at +2 volts by a resistor network. When the other input is lower than 2 volts, the output will swing high; any input above 2 volts causes the output to switch low.

Figure 3A is an example of using a switch to generate an input signal for U2. This simple design has a major flaw that ignores an innate characteristic of any mechanical switch. When the contacts of a mechanical switch come together they invariably bounce, making and breaking the contact several times. Computers and other electronic circuits are fast enough to react to each bounce, giving the effect of several switch closures. Consequently, the additional components shown in figure 3B are used to prevent the multiple closures from generating multiple signals. Resistor R1 requires a long time to charge capacitor C1; when the switch discharges C1 and then bounces open, the voltage input to U2 cannot change fast enough to cause U2 to switch. Even if the switch bounces a number of times before staying closed, only one signal is generated. When the switch opens, any bounce will have died away before U2's input can change, and again only one signal is generated. R2 limits the discharge current to protect the contacts.

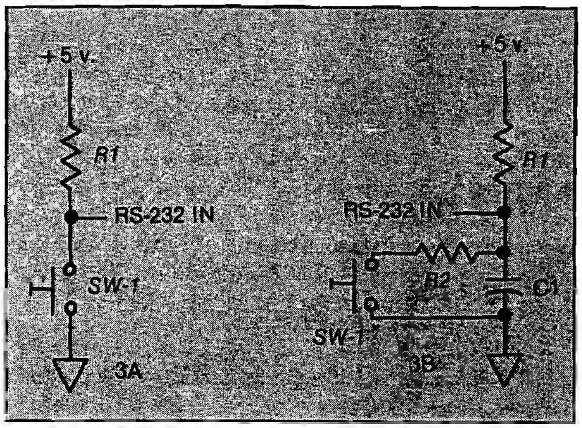


Figure 3. **Switch inputs for the RS-232 IN line. In 3A, a naked switch can generate multiple input level transitions as the mechanical contacts bounce during operation. Additional components in 3B prevent switch bounce from generating more than one closure signal.**

Figure 4 shows a LED (light emitting diode) driven by a single transistor; the circuit design allows us to safely drive the transistor with almost 24 volts of signal (+ and −12 volts) from CoCo's serial output line. When the transistor is turned on, current will flow through the LED and transistor, lighting the LED. When the serial output line goes high (RS-232 logic 0), the transistor turns on. Diode D1 comes into play when the signal swings negative (RS-232 logic 1). D1 is forward biased (current flows in the direction of the arbow symbol), which allows it to absorb the current generated by the input signal. This is a necessary precaution for many transistors, since their input may not be able to withstand the −12 volts reverse voltage that would otherwise be applied to the transistor.

To use the interface we have designed, it will be necessary to program the internal PIA so that it will respond to our signal (switch closure) and to turn on the transistor LED driver. During system reset, all PIA internal registers are set to zero, which programs all PIA I/O lines as inputs. As CoCo boots up into operation, the PIA registers are initialized so that each I/O line is set up for its assigned task. For this experiment, we will use the same initialization and we can skip a step normally required to use a PIA. To turn on the transistor, we must set the RS-232 line high (RS-232 logic 0). This is done by setting the PIA output line to digital logic 0. Similarly, to detect a switch closure, we must read the RS-232 input line on the PIA. Our circuit design causes a logic 1 to appear on the port pin when the switch is closed.

Since the PIA is controlled by the processor data bus, each I/O transaction involes eight bits; we will have to sort out the single input or output bit associated with our circuit. In figure 1 note that PA1 drives the output amplifier, while PB0 is driven by the input amplifier. Port 1 (PA0-PA7) is located at $FF20, and Port 2 (PB0-PB7) is located at $FF22.

To separate those two port lines, we must use the Boolean operators AND and OR if we program in BASIC. For example, the following program will alternately turn on and turn off line PA1 of the PIA;

```
10 A = PEEK(65312):C = A AND 254:
   POKE(65312),C

20 B = PEEK(65312):C = B OR 2:POKE
   (65312),C

30 GOTO10
```

For those unfamiliar with these instructions, the BASIC AND instruction performs the Boolean AND operation using these Boolean statements: 1 AND 1 = 1, 1 AND 0 = 0, as illustrated here.

	10101100
AND "Mask"	11110000
Result	10100000

This simple problem shows the characteristics of logical AND operations. First, the operand for the AND is

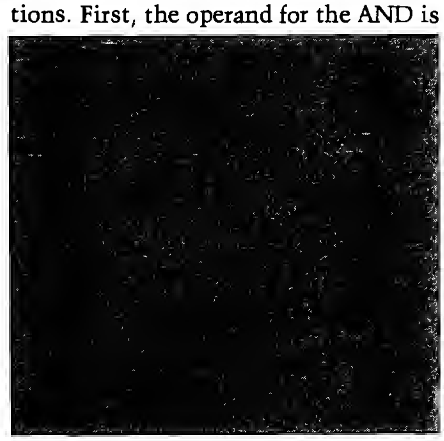


Figure 4. **LED driver circuit to receive RS-232 level signals from the RS-232 OUT line. Diode D1 protects transitor base from excessive reverse voltage when Rs-232 signal swings to −12 volts for logic one signal.**

called a mask; in hex notation, the mask used above is $F0. The name mask comes from the fact that the all-ones nibble (half a byte) protected the corresponding four bits, while the all-zeros nibble blanked out the four bits in the low-order nibble. We say that 0 bits turn *off* corresponding bits in the value being operated on. Similarly, the OR instruction turns *on* corresponding bits in a binary word:

```
               10101100
OR "Mask"      11110000
               --------
Result         11111100
```

Re-read the BASIC program example and note that line 10 reads the value in location 65312, turns *off* PA1 without changing *any other bit*, then rewrites the value. Line 20 reads the value, turns *on* PA1, and restores the word to the port. It is often important to change only one bit of an I/O port at a time, and this program is one way to do it.

An assembly-language program that does the same thing follows:

```
START LDA  $FF20 READ THE
                 LOCATION
      ANDA #$FD  TURN OFF BIT 1
      STA  $FF20 REPLACE VALUE
      LDA  $FF20 READ VALUE
                 AGAIN
      ORA  #$02  TURN ON BIT 1
      STA  $FF20 REPLACE VALUE
      BRA  START LOOP FOREVER
```

On the input side, the following BASIC program will test PB0 of the port (RS-232 IN line):

```
10 A=PEEK(65314):B=A AND 254:
   IF B=0 THEN 30
```

With these programming examples, our interface design in figures 3 and 4 can come alive. The circuits can be built using any construction techniques you wish; the layout is not critical. A list of suggested parts is included below. Although this is a rudimentary design, it is one type of interface applicable to many home computers that are capable of driving standard printers. Future examples will expand this single-bit port to multi-bit operation, and suggest some support circuitry.

The circuit shown is quite safe for the computer as long as it is wired as shown. Special precautions: do not expose any pin of the serial port to voltages higher than 6 or 7 volts; do not expose pin 4 to a voltage input at all. Similar statements can be made about any other computer, with suitable modification regarding pin numbers of the port connector.

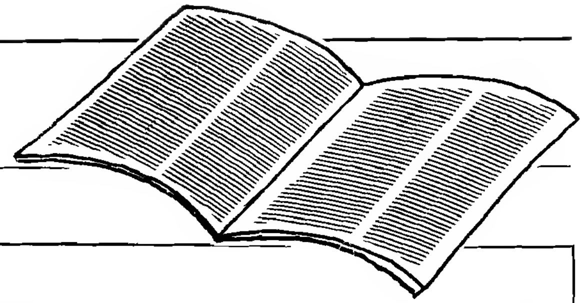
Parts List for Single-Bit Serial I/O

R1 - 4.7K (4700) ohm ¼ w. resistor, R/S p/n 271-1330
R2 - 10 ohm ¼ w. resistor, R/S p/n 271-1301
R3 - 10K (10000) ohm ¼ w. resistor R/S p/n 271-1335
R4 - 22 ohm ¼ w. resistor (use 2 R/S #217-1307 in parallel)
C1 - 22 μF 16-volt capacitor, R/S p/n 272-1426
SW-1 push button switch, R/S p/n 275-1547 or 276-618, 276-1549
LED (any color) R/S p/n series 276-041 — 276-071
D1 - 1N914 or 1N4148, R/S p/n 276-1122
Cable serial port - R/S p/n 26-3020 (makes two; cut in half)
Breadboard - R/S p/n 276-170 or 276-158

Please forward questions and suggestions for discussion topics to Mr. Tenny at P.O. Box 545, Richardson, TX 75080.

MICRO

MICRO™



Annual Index

Annual Index *(continued)*

EDUCATION

GAMES

GRAPHICS

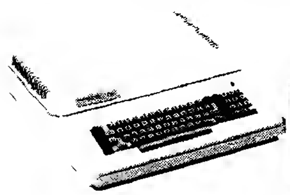
Annual Index *(continued)*

HARDWARE

(MICRO's Annual Index will be continued next month.)

computer mail order

FRANKLIN



Call on FRANKLIN Computers, Disk Drives, Software and Special System Offers!

MICRO-SCI
DISK DRIVES FOR APPLE & FRANKLIN

A2	$269 00
A40	$349 00
A70	$459 00
C2 Controller	$79.00
C47 Controller	$89 00

RANA DISK DRIVES

Elite I plus (Apple/Franklin)	CALL
1000 (Atari)	$349 00

SYSCOM
48K Color Computer
APPLE COMPATIBLE
$629

VISICORP
for Apple, IBM & Franklin

Visidex	$189 00
Visifile	$189 00
Visiplot	$159 00
Visiterm	$89 00
Visitrend/Plot	$229.00
VisiSchedule	$229 00
Desktop Plan	$189 00
Visicalc (Apple II, CBM, IBM)	$179.00

Visicorp prices for IBM may vary slightly

BRODERBOUND

Apple Panic	$23 00
David's Magic	$27 00
Star Blazer	$25 00
Arcade Machine	$34 00
Choplifter	$27 00
Serpentine	$27 00

INFOCOM

Deadline (Apple, Atari, IBM)	$35.00
Star Cross	$29 00
Zork I, II or III	$29 00

MPC

Bubdisk (128K Non Volitare)	$649 00

AXLON

Apple/Franklin 128K Ram	$399 00
Apple/Franklin Ram Disk	$999 00

KRAFT

Apple Joystick	$44 00

CONTINENTAL

Tax Advantage (Apple/Atari)	$45.00
Home Accnt (Apple/Atari)	$59 00
1st Class Mail/Form Letter (Apple)	$79 00
The Book of Apple	$14.95
The Book of Atari	$14.95
The Book of Apple Graphics	$14 95

PERCOM

DISK DRIVES FOR ATARI

AT 88-S1	$399 00
AT 88-A1	$299 00
AT 88-S2	$649 00
RFD 40-S1	$549 00
RFD 40-A1	$349 00
RFD 40-S2	$889 00
RFD 44-S1	$679 00
RFD 44-S2	$999 00

FLOPPY DISKS

MAXELL

MD I (Box of 10)	$32 00
MD II (Box of 10)	$44.00
FD I (8")	$40 00
FD II (8" DD)	$50.00

VERBATUM

5¼" SS DD	$26 00
5¼" DS DD	$36 00

ELEPHANT

5¼" SS SD	$19.99
5¼" SS DD	$24.99
5¼" DS DD	$29 99

HEAD

Disk Head Cleaner	$14.95

SIRIUS

Bandits (Apple)	$28.00
Beer Run (Apple)	$24 00
Free Fall (Apple)	$24.00
Sneakers (Apple)	$24.00
Snake Byte (Apple)	$24 00
Fast Eddie (Atari)	$21 00
Turmoil (Atari)	$21.00
Deadly Duck (VIC)	$21.00

INTERFACES & ACCESSORIES

80 Col Apple Card	$159 00
Apple Parallel Printer Interface	$69.00
Atari 850 to Parallel Printer Cable	$29 00
RS232—RS232 Cables	$29 00
Centronics Parallel—Parallel Cables	$29 00

Call on IBM, Osborne, Daisywriter, Atari, Commodore, Apple and Franklin Interface Cards, Cables and accessories

HEWLETT PACKARD

41CV
$209

HP 75 .... $749.

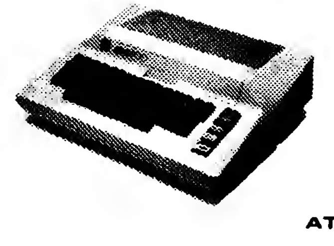
HP 41C (Free Memory Module)	$149 00
HP 10C	$59.00
HP 11C	$72 00
HP 12C	$99 00
HP 15C	$99 00
HP 16C	$99 00

For HP 41/41CV

HPIL Module	$99.00
HPIL Cassette or Printer	$359 00
Card Reader	$144 00
Extended Functions Module	$64.00
Time Module	$64 00
Mathmatics Pac	$26.50
Real Estate Pac	$40.00

HOME COMPUTERS

ATARI



ATARI 800
48K
$489

ATARI 400

16K $199
32K $274*
48K $299*
64K $359*

*Non-Atari Ram

ATARI 600-16K ATARI 1200-64K

Call for price and availability on these two new Computers from Atari.

810 Disk Drive	$429.00
Inhome Keyboard for Atari 400	$89.00
1010 Program Recorder	$74.00
1020 40 Col. Printer/Plotter	$269.00
1025 80 Col. Printer	$469.00
830 Acoustic Modem	$159.00
850 Interface Module	$169.00
CX40 Pr. Joysticks	$16.00
CX418 Home Manager Kit	$69 00
CX419 Bookkeeper Kit	$195.00
CX482 Educator Kit	$129.00
CX483 Programmer Kit	$54 00
CX488 Communicator Kit	$229.00
CX7101 Entertainer Kit	$69 00
Invitation to Programming I	$18.00
Invitation to Programming II	$20.00
Invitation to Programming III	$20 00
4002 Basic Language	$42.00
4003 Assembler Editor	$47.00
8121 Micro Assembler	$69.00
8126 Micro Soft	$69.00
405 Pilot (Edu.)	$105.00
4018 Pilot (Home)	$72.00
8036 Atari Writer	$79.00
404 Word Processor	$119 00
5059 Visicalc	$159.00

ATARI

Pacman	$33.00
Centipede	$33.00
Defender	$33 00
Galaxian	$33.00
Missile Command	$29 00
Star Raiders	$33.00
Caverns of Mars	$32.00
Dig Dug	$33.00
Donkey Kong	CALL
E.T. Phone Home	$39.00
Eastern Front (1941)	$39.00
QIX	$33.00
Superman III	$39.00
Star Trux	$33.00
Asteroids	$29 00
Basketball	$29 00
Computer Chess	$29 00
Juggles House	$23.00
My First Alphabet	$29.00

APX

Text Formatter	$18.50
Family Budgeter	$18.50
Eastern Front	$24 00
Family Cash	$18.50
Jukebox	$13.50
Downhill	$18 50
Outlaw	$18 50
Holy Grail	$24.00
Player Piano	$18 50
Keyboard Organ	$18 50
Number Blast	$13.50
Frogmaster	$18.50
747 Land Simulator	$18 50
Bumper Pool	$13.50

ON-LINE

Jawbreaker	$27 00
Softporn	$27.00
Wizard and Princess	$29 00
The Next Step	$34.00
Mission Asteroid	$22 00
Mouskattack	$31 00
Frogger	$31.00
Cross Fire (ROM)	$36.00

Some ON-LINE Products are also available for Apple and IBM. Call for information.

SYNAPSE

File Manager 800 plus	$69.00
Chicken (ROM)	$34.00
Picnic Paranoia (ROM)	$34 00
Claim Jumper (ROM)	$34.00
Slime (ROM)	$34.00
Shamus (ROM)	$34 00
Protector (ROM)	$34 00
Dodge Racer (C/D)	$26.00
Nautilus (C/D)	$26.00
Shadow World (C/D)	$26.00
Survivor (C/D)	$26 00
Drelbs (C/D)	$26.00
Necromancer (C/D)	$26 00
Pharoh's Curse (C/D)	$26 00
Fort Apocolypse (C/D)	$26 00
Page 6	$19.00
Assembler	$30 00
Disk Manager	$24.00

DATASOFT

Pacific Coast Highway	$25 00
Canyon Climber	$25 00
Tumble Bugs	$25 00
Shooting Arcade	$25 00
Clowns and Balloons	$25 00
Graphic Master	$30 00
Graphic Generator	$13 00
Micro Painter	$25 00
Text Wizard	$79.00
Spell Wizard	$64 00
Bishop's Square	$25 00
Sands of Egypt	$25.00
Moon Shuttle	$25.00
Zaxxon	$29 00

ALIEN

Atari Voice Box	$119 00
Apple Voice Box	$149 00

MEMORY

Axlon 32K Ram	$89 00
Axlon 48K Ram	$139.00
Axlon 128K Ram	$399 00
Intec 32K Board	$74 00
Intec 48K Board	$99 00
Intec 64K Board (400 only)	$149 00

WICO

Joystick	$24 95
Famous Red Ball	$26.95
Apple Trackball	$59 00
Atari/VIC Trackball	$55.00
Apple Adapter	$16 00

CBS

K-razy Shoot Out	$32 00
K-razy Kritters	$32 00
K-razy Antics	$32 00
K-razy Patrol	$32 00
Stick Stand	$5 99

EPYX

Crush, Crumble & Chomp	$24 00
Crypt of the Undead	$24 00
Curse of Ra	$16 00
Datestones & Ryn	$16 00
Invasion Orion	$19 00
King Arthur's Heir	$24 00
Morloc's Tower	$16 00
Rescue at Rigel	$24 00
Ricochet	$16 00
Star Warrior	$29 00
Temple of Apshai	$29 00
Upper Reaches of Apshai	$16 00

SPINNAKER

Snooper Troops #1	$34 00
Snooper Troops #2	$34.00
Face Maker	$24 00
Story Machine	$24 00
Delta Drawing	$45 00
Rhymes and Riddles	$21 00
Kinder Comp	$21.00

ROKLAN

Wizard of War (ROM)	$34 00
Deluxe Invader (ROM)	$29.00
Gorf (ROM)	$34 00

FIRST STAR

Astro Chase	$25 00

BIG 5

Miner 49er	$35 00

GAMESTAR

Baja Buggies	$24 95
Football	$24 95

SUBGECO.

Raster Blaster	$24 95

L.J.K.

Letter Perfect 40/80 Col Disk	$109 00
Letter Perfect 40 Col Rom	$179 00
Letter Perfect 80 Col. Rom	$179 00
Data Perfect 40/80 Col Disk	$79 00
Mail Merge	$21 95

CALL FOR APPLE/L.J.K. PRODUCTS

BIT 3

80 Col Card for Atari 800	$279 00

We also stock software by: Adventure International, Eduware, Creative Software, Dorsett Educational, IDSI, Romox and Artworx. Call for details and prices.

computer mail order west

computer mail order east

800-233-8950

IN PA. CALL (717)327-9575

DEPT 415, 477 E. Third St. Williamsport, PA 17701

No risk, no deposit on C.O.D. orders. Pre-paid orders receive free shipping within the UPS Continental United States Delivery Zone, with no waiting period for certified checks or money orders. Add 3% (minimum $3.00) shipping and handling on all C.O.D. orders. Larger shipments may require additional charges. NV and PA residents add sales tax. All items subject to availability and price change. NOTE: We stock manufacturer's and third party software for most all computers on the market. Call today for our new catalog.

MICRO™ Reviews

Product Name: **HUMBUG**
Equip. req'd: TRS-80 Color Computer with 16K
Price: $39.95 on cassette
Manufacturer: STAR-KITS
P.O. Box 209
Mt. Kisco, NY 10549

Description: *HUMBUG* is an exceptionally complete and well-written assembly-language debug monitor for the Color Computer. It requires 4K of read/write memory anywhere in the machine and is totally position-independent. *HUMBUG* can reside in ROM, and then uses approximately 60 bytes of read/write memory for workspace and 256 bytes for two separate stacks. Almost any screen display can be printed at any of five selectable baud rates. Memory can be dumped in ASCII, hex, or disassembler format, as well as a continuous ASCII text string. Other memory commands allow video display of any 512-byte section of memory, memory examine/change, memory compare, and test memory. *HUMBUG* will save assembly-language programs on tape, but the major input/output mode is an RS-232 data stream in the Motorola checksum S1-S9 format. The debug features include single-step and breakpoint operation and register examine/change.

Pluses: The "desembler" command is not a full disassembler, but it dumps memory in disassembly format to ease code entry. The FIND command prints the byte previous to and two bytes following the specified byte. For example, FIND $16 will list every LBSR command in the program being analyzed — by location! The ANALYZE TAPE command reports the start and end addresses, the transfer address, and record type of any block-recorded tape. Finally, MOVE HUMBUG moves *HUMBUG* anywhere in read/write memory, and WHERE HUMBUG reports the location of the current version.

Minuses: *HUMBUG* does not read machine-language tapes, and the documentation on I/O control needs step-by-step examples. Also, the listing was photo-reduced, then part of each comment was truncated to make room on the page.

Documentation: A 30-page book provides a full listing and covers operation of *HUMBUG*, memory requirements, compatibility with BASIC, and protection of HUMBUG when BASIC is running. This documentation is well done and very thorough.

Skill level required: Experienced assembly-language programmers will derive the most benefit from this program, but the abundance of commands and excellent documentation will help novices learn assembly-language programming.

Reviewer: Ralph Tenny

Editorial Comment: This reviewer commends STAR-KITS for giving a full listing and for facing the software piracy

(Continued on page 112)

Reviews *(continued)*

issue head-on. In effect, they say "Here's how to copy HUMBUG; don't give copies away or we will sue you!" By furnishing the listing and very complete documentation, STAR-KITS has made it possible for the user to derive the fullest value possible from his investment, and the resulting product is thereby more durable and valuable.

Product Name:	**Readtest**
Equip. req'd:	64K TRS-80 Color Computer, one disk drive, FLEX DOS
Price:	$54.95, $74.95 with source
Manufacturer:	Frank Hogg Laboratory 770 James St. Syracuse, NY 13203
Author:	Dale Puckett

Description: *Readtest*, a utility for writers, is a text analysis program that reads standard FLEX text files from disk and generates a written report on your terminal or printer. The program gives you the number of words, lines, sentences, pronouns, proper nouns, and affixes in your text, and also computes the average sentence length. The report rates your material from very difficult to very easy to read, and classifies it from dramatic to dull. You are told what percentage of persons in the U.S. could read your material and what grade-level reading skill is required to understand it. The program is based on Dr. Rudolf Flesch's book, "The Art of Plain Talk," and a statistical readability formula.

Pluses: The report display conforms to your terminal standards automatically and wraps words on the screen. The program-generated report is easy to understand. *Readtest* is not copy-protected and takes up only 24 256-byte sectors on a FLEX-formatted disk. It is written in position-independent code and can be loaded and run anywhere in memory.

Minuses: Although *Readtest* is written in assembly language, it can take several minutes to generate a report with a 25000-word text file. *Readtest* runs under the FLEX operating system only and you need a standard FLEX editor to generate text for the program.

Documentation: The author has prepared a thorough, comprehensive, and easy-to-read manual that details each command, explains how the program works, how to adapt it to other systems, and how to effectively interpret the results of the printed report. The documentation is over 13 pages long. Even if you don't buy the source code, *Readtest's* routines are explained in detail. A demonstration text file is included to show how *Readtest* works and how not to write.

Skill level required: No special techniques or abilities required.

Reviewer: Bill Ball

Product Name:	**HESMON — VIC Monitor Cartridge**
Equip. req'd:	VIC-20 (5K or more)

Price: $39.95
Manufacturer: Human Engineered Software
71 Park Lane
Brisbane, CA 94005
Author: Dr. Terry Peterson

Description: *HESMON* is a 6502 machine-language monitor for the VIC-20 similar to the various versions of MICROMON or SUPERMON available for the PET. *HESMON* comes on a cartridge that plugs into the VIC's expansion port. It has nearly 30 commands — almost double the number available in Commodore's VICMON. In addition to the standard commands (e.g., A,B,C,D,F,G,H,I,J,L,M,N,Q,R,S,T,W, and X) found in VICMON, *HESMON* has hex-to-decimal conversion, hex arithmetic, a memory test, a color test, and an external relinker that enables you to convert code from one machine to another if you have a table of corresponding memory locations. Output may be optionally diverted to a printer or disk. Both forward and reverse scrolling are implemented.

Pluses: *HESMON* is a great value — nearly twice the commands of VICMON for only two-thirds the price ($39.95 *versus* $59.95). It is completely compatible with BASIC, and there is no need to protect zero-page as with VICMON.

Minuses: *HESMON* will not disassemble itself, or allow you to look at it or transfer it in any manner. This is unfortunate because there is much that could be learned from this fine implementation.

Documentation: The 32-page manual is clearly written and contains many examples. The first section, entitled "If You've Never Used a 'Machine-Language Monitor' Before," is quite unique and well done.

Skill level required: Basic understanding of 6502 machine language.

Reviewer: David Malmberg

Product Name: **Dual Plug-in Interface Board, p/n 276-163**
Equip. req'd: TRS-80 Color Computer, breadboarding tools, and materials as required for project
Price: $4.95
Manufacturer: Radio Shack

Description: Although Radio Shack did not intend that p/n 276-163 be used with the Color Computer, and the documentation does not suggest such a use, this product is a low-cost source for prototyping and expansion boards for the Color Comptuer. It is a 4.25" × 5.25" glass epoxy board with a 40-pin card-edge connector on either end. Both ends are a precise mechanical and electrical fit for the computer's expansion port. The board will hold 15 normal-sized ICs or varying numbers of larger ICs, including .6"-wide ICs. There is also a modest space for discrete components and two power bus lines are furnished.

(Continued on next page)

Pluses: Low cost and good construction allows near-professional interface/expansion for the Color Computer. A longer card has also been announced.

Minuses: None noted.

Documentation: One page listing part numbers of Radio Shack connectors and accessories compatible with the board, plus construction and maintenance hints.

Skill level required: It should be stressed that this product must *not* be used in the expansion port of the Color Computer unless the user is fully aware of computer interfacing techniques, and that damage caused by improper use of the product could void any existing warranty. Users with the proper experience still must exercise proper care to avoid damage to the computer.

Reviewer: Ralph Tenny

Product Name:	**Snooper Troops — The Granite Point Ghost and the Disappearing Dolphin**
Equip. req'd:	Apple II with Applesoft or Apple II Plus with 48K RAM, DOS 3.3
Price:	$44.95
Manufacturer:	Spinnaker Software 215 First St. Cambridge, MA 02142
Author:	Tom Snyder
Copy Protection:	Yes

Description: *Snooper Troops* is a child's educational program designed to look like a game while teaching how to think logically. It also develops skills in note taking, map drawing, and information classification. The object of the program is to discover the who, what, where, and when, using appropriate detective skills.

Pluses: *Snooper Troops* is well written and challenging. It will keep a child occupied for weeks attempting to track down all the clues. Additional copies (you receive two) are available for $2.00 from the company.

Minuses: The graphics are interesting for about five minutes. Each house you travel to looks identical, leading to loss of excitement. The program incorporates some lengthy delays that I find annoying.

Documentation: Well written, informative guide with spaces for note taking.

Skill level required: While the company recommends ages ten to adult, I think most adults would be bored after an hour or so. I recommend 8 to 12.

Reviewer: Phil Daley

MICRO™

Software Catalog

Classroom Help

Computerized Gradebook for the Apple assists the classroom teacher in storing, retrieving, and computing grades in a systematic manner. The program will store fifty student names, with up to ten scores per student. It will translate raw scores into standard scores, weight scores by percentage, compute total weighted scores, and assign letter grades.

The **$49.95** price includes one diskette and user instructions. Available from Electronic Courseware Systems, Inc., P.O. Box 2374, Station A, Champaign, IL 61820; (217) 359-7099.

CoCo Screen Enhancer

The **64K Screen Expander** allows the 64K Color Computer to have a 51 x 24 upper- and lowercase display. This includes BASIC and all assembly language programs that use text displays. A character editor lets you change any of the characters.

Price is **$24.95** for cassette and **$29.95** for disk (plus $2 shippng and handling). Available from COMPUTERWARE dealers or directly from COMPUTERWARE, Box 668, 4403 Manchester Ave., Suite 103, Encinitas, CA 92024; (619) 436-3512.

Making Math Fun

The VIC-20 **MicRo Math Blaster** is a combination of arcade game excitement and basic mathematic drill in addition, subtraction, multiplication and division. Color graphics and sound combine to motivate the child to practice these skills. Levels of difficulty range from grades 1-8.

Price is **$15.95**. Available from M-R Information Systems, Inc., P.O. Box 73, Wayne, NJ 07470.

VIC-20 Space Game

Martian Raider will have you streaking through deep space as you guide your intergalactic battle ships in an attack on Mars. You skim perilously close to the surface of the planet, devastate Martian cities and destroy ammunition dumps, while warding off ground-to-air missiles, U.F.O.'s, and meteorites. Martian Raider requires an unexpanded VIC-20 and cassette player; keyboard or joystick controlled.

Price is **$19.95**. Available from your computer dealer or Broderbund Software, Inc., 1938 Fourth Street, San Rafael, CA 94901; (415) 456-6424.

Spelling Check

Sensible Speller, spelling verification program for the Apple, is now available in a formatable version. This makes it possible for the individual with more than one type of word processor to utilitze the same speller regardless of the type of file the word processor generates. Sensible Speller still features the complete Concise Edition of The Random House Dictionary on disk and hardcover book, over 80,000 words in all. The new Speller works with DOS 3.3, DOS 3.2, CP/M, Pascal, Word Handler, and Super Text.

Sensible Speller requires an Apple II/Apple II+ with 48K, DOS 3.3, and one or two disk drives. Two disk drives are required to delete or add words to the dictionary.

Price is **$125.00**. Available from Sensible Software, 6619 Perham Drive, W. Bloomfield, MI 48033; (313) 399-8877.

3-D Astronomy

Celestial Simulator provides 3-dimensional educational Science software for the Apple. Programs show moving 3-D simulations of the planets revolving around the earth as the earth revolves around the around the earth as the earth revolves around the sun from different positions in the universe. Watch a 3-D simulation of a solar and lunar eclipse. See the moon's shadow follow its movement on the earth.

Observe the stars revolving at night. Have the stars plot their paths. Give any planet's orbit an eccentricity and watch it revolve around a star while plotting its path. Take a simulated flight to Alpha Centauri — control your acceleration, speed, weight. Try not to crash into Alpha Centauri when you reach it. Compare time using Einstein's theory of relativity.

Your imagination can run wild making solar systmes change and causing rockets to blast off or planets to crash or freeze.

Celestial Simulator runs on a 48K Apple II/II+ w/Applesoft in ROM or a Language Card and DOS 3.3. Price is **$99.95.** Available from Soft Images, 200 Route 17, Mahwah, NJ 07430; (201) 529-1440.

Auto Racing on the CoCo

Revolution for the Radio Shack Color Computer and the TDP-100 uses a series of conversational screens and menus to simulate the experience of being a race driver. Discussions with the team, choosing cars and tracks to race on, and receiving information on lap times and lap records are all part of the game.

Revolution is menu-driven and uses single-key entry wherever possible. 32K disk price is **$24.95**; cassette price is **$22.95**. Contact Inter+Action, 113 Ward St., New Haven, CT 06519; (203) 562-5748.

Adventure Game

ZORK III, a prose adventure game, takes players into a great underground empire where they encounter the Dungeon Master, who holds their fate in the balance. The game has a subterranean fantasy theme. Zork III can be used with Apple, Atari, IBM, Radio Shack TRS-80, Commodore, NEC, Osborne, CP/M, and DEC computers.

Price is $39.95 (NEC-APC, CP/M, PDP11 Models **$49.95**). Available from Infocom, Inc., 55 Wheeler St., Cambridge, MA 02138; (617) 492-1031.

Electronic Spreadsheet

NOVACALC, a full-featured spreadsheet/financial planner, has all the

(Continued on next page)

standard spreadsheet functions like math, trig, log, sum, and average functions along with screen controls. Its advanced features include functions for depreciation (DEP), net present value (NPV), rates of return (IRR), true consolidation or roll-up (RUP) for three-dimensional spreadsheet operations. NOVACALC also features decimal and comma format control, selective column printing, wide carriage printing, helping menu and help screen, underlining, protected data fields and sharing of data among worksheets to avoid data reentry.

NOVACALC is available for the Apple (with Z-80 card), Cromemco, IBM-PC, Kaypro, Northstar, Osborne (D.D.), Televideo, Vector, Xerox and 8" CMP. Price is **$24.95**. For more information contact Hourglass Systems, P.O. Box 312, Glen Ellyn, IL 60137; (312) 690-1855.

Data Base Management

Version 2.0 of FILE-FAX Data Base Management System is designed as a filing system for the businessman who wants a program that is simple to learn and easy to operate. FILE-FAX offers quick access to files and records, as well as an easy-to-use report generator, according to TMQ Software, manufacturer.

Among its applications are inventory control, customer files, mailing lists, purchase records, patient billing, and salesmen territory files.

FILE-FAX will run on Apple II, Apple II+ and IIe computers (**$149.00**), as well as the Atari 800 (**$129.00**). For more information contact TMQ SOFTWARE, INC., 82 Fox Hill Drive, Buffalo Grove, IL 60090; (312) 520-4440.

Apple Word Processor

Screenwriter II for the Apple II/II+ offers global search and replace, proportional spacing, insert and delete and text move, true upper- and lower-case text on your screen, 70-column displan on screen, and text spooling. Requires 48K.

The **$129.95** price includes two master disks, documentation, and reference cards. Available from Sierra On-Line Inc., Sierra On-Line Building, Coarsigold, CA 93614; (209) 683-6858.

Apple Software for Bank Security Officers

Two new software packages — **Loss Control** and **Audit Control**, are designed to reduce the costly, time-consuming, and error-prone manual procedures required by bank security officers and auditing personnel.

Loss Control package allows security personnel to track and analyze bank losses and recoveries by operating unit and category. Users can inquire into loss history using up to 13 different search criteria. They can also compare selected categories of current operating losses with prior months and years to spot problem trends and take corrective action.

Audit Control streamlines the production of audit reports and ensures accurate reporting and analysis of findings. The package combines the separate audit functions of data entry, reporting, and follow-up into one procedure through the use of the Audit Control "Turnaround" report. Descriptions of standard findings, as well as often-used text, can be saved and then called up repeatedly as required.

Audit Control, priced at **$750**, is available for the Apple II, Apple III, and IBM PC. Loss Control, also **$750**, is available for the Apple II Both packages require a minimum of 64K bytes of RAM and dual floppy disks. The packages include program diskettes, initial data diskettes, a self-teaching demo, and a user manual.

Available from Quest Designs, Inc., 371 22nd Street, Oakland, CA 94612; (415) 839-1579.

Graph Program

Sidebar for the VIC-20 enables you to construct, edit, and store on tape titled bargraphs. you can quickly review a series of graphs stored on tape, plus edit and re-store them.

Price is **$9.60**. Available from MFJ Electro-Enterprises, P.O. Box 13076, Kanata, Ont. K2K 1X3 Canada; (613) 592-2962.

MICRO

Communications Interface

Asynchronous Interface Adapter for Printers and Communication

The requirement to get information out of a microcomputer to a printer, another microcomputer, or *via* the telephone to any remote system, is basic to many microcomputer systems. An industry standard, the 'RS-232', is useful in most of these applications. The RS-232 is a standard for 'serial' communication. This means that information is broken down into its lowest form — a string of ones and zeros — transmitted one bit at a time. The common type of modem is called 'asynchronous' and adheres to the RS-232 protocol. Unfortunately, this very general and useful standard does not come as part of the basic equipment of many popular systems, including the Apple II, PET and other CBM systems, AIM, SYM, or KIM, and other single board computers, and so forth. Fortunately, there is an easy way to add the RS-232 capability to your system.

The Synertek 6551 Asynchronous Communication Interface Adapter is a very intelligent chip that performs baud-rate generation, parity generation and checking, full-duplex or half-duplex operation, error detection, and more. It can be used with the Motorola 1488 and 1489 to provide a full RS-232 standard set of signals to control printers, modems, and communication between intelligent devices.

Pin Configuration

GND	1	28	Read/Write
CS_0	2	27	Phase 2
CS_1	3	26	IRQ
Reset	4	25	Data Bit 7
RxC	5	24	Data Bit 6
XTAL1	6	23	Data Bit 5
XTAL2	7	22	Data Bit 4
RTS	8	21	Data Bit 3
CTS	9	20	Data Bit 2
TxD	10	19	Data Bit 1
DTR	11	18	Data Bit 0
RxD	12	17	DSR
RS_0	13	16	DCD
RS_1	14	15	Vcc

Description of Signals Between the ACIA and the Microprocessor

CS_0 and CS_1 are Chip Selects generated by the microprocessor circuitry to select the ACIA 6551.

Reset is the master system Reset, which establishes initial conditions within the ACIA 6551.

RS_0 and RS_1 are Register Selects, normally address lines A_0 and A_1, and are used to select the four internal addresses of the ACIA 6551.

DB_0 through DB_7 are Data Bus lines used to transfer data between the microprocessor and the ACIA.

IRQ is the system Interrupt ReQuest.

Phase 2 is the system clock.

Read/Write is the system data direction signal.

Between the ACIA and the External World

RxC is a Receive Clock, which can be used to generate or receive external clocking for the receiver portion of the ACIA.

$XTAL_1$ and $XTAL_2$ normally connect to a 1.8432 megahertz crystal to provide the baud-rate generation.

RTS (Ready To Send) is an output pin used to control the Ready To Send pin of the modem or other device.

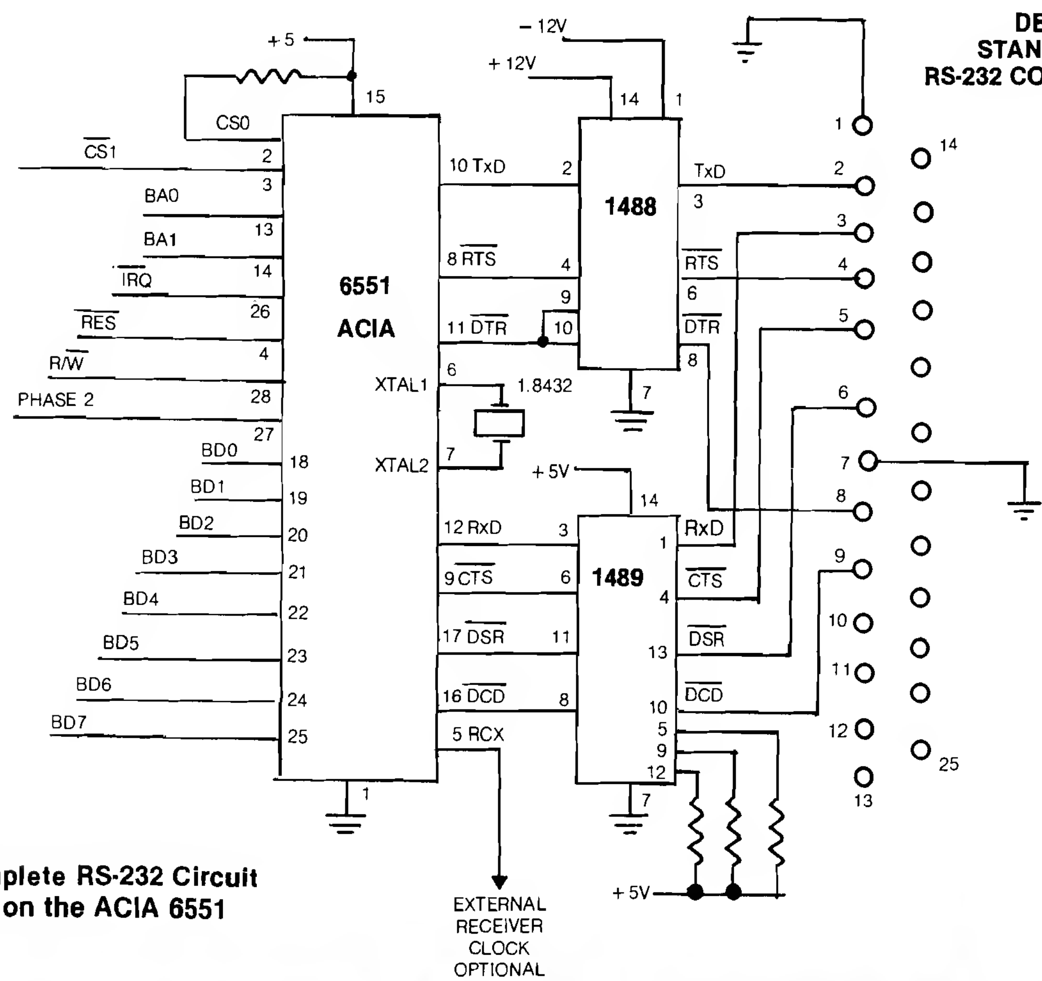
CTS (Clear To Send) is an input pin used to detect when the modem or other device is ready to have data transmitted to it.

TxD is the output line used to send data from the Transmit Register of the ACIA serially (one bit at a time) to the modem.

DTR (Data Terminal Ready) is an output pin that indicates to the modem when the ACIA is enabled.

DCD (Data Carrier Detect) is an input pin that indicates to the ACIA when the modem has a carrier signal.

DSR (Data Set Ready) is an input pin that indicates the status of the modem.



A Complete RS-232 Circuit Based on the ACIA 6551

Communications Interface

A Complete RS-232 Circuit Based on the ACIA 6551

The RS-232 standard permits a wide range of values to represent the high and low signals. The ACIA 6551 provides only TTL-level signals. This may be adequate for specialized applications such as talking to a printer, but many modems and some printers require the higher voltage levels, ranging up to about ±12 volts. The ACIA and crystal provide all that is required for a TTL-level system. The 1488/1489 devices provide buffering and signal levels required to meet the complete RS-232 specification. This may be required for many standard modems. It does not add much complexity or expense to the basic circuit.

ACIA 6551 Registers

Data Register — Register 0
Write to Register 0 will transmit data.
Read from Register 0 will receive data.

Status Register — Register 1

Bit	Hex	Function
0	01	1 — Parity Error on Receive
1	02	2 — Framing Error on Receive
2	04	4 — Overrun Error on Receive
3	08	8 — Receive Data Register Full
4	10	10 — Transmit Data Register Empty
5	20	20 — Not Data Carrier Detect
6	40	40 — Not Data Set Ready
7	80	80 — Interrupt Request

Command Register — Register 2

Bit	Hex	Function
0	01	0 — Disable Receiver and Interrupts
		1 — Enable Receiver and Interrupts
1	02	0 — IRQ Enabled from Receiver Data Full
		IRQ Disabled
2	04	00 — Xmit Int. Disabled / RTS High / Xmit Off
3	08	04 — Xmit Int. Enabled / RTS Low / Xmit On
		08 — Xmit Int. Disabled / RTS Low / Xmit On
		0C — Xmit Int. Disabled / RTS Low / Xmit BRK
4	10	0 — Normal — No Echo
		1 — Echo (Bits 2,3 must be 0)
5	20	00 — Parity Disabled
6	40	20 — Odd Parity
7	80	60 — Even Parity
		A0 — Mark Parity Xmit, Recv. Parity Off
		E0 — Space Parity Xmit, Recv. Parity Off

Control Register — Register 3

Bit	Hex	Function
0	01	Baud Rate: 00 — 16 times External Clock
1	02	01 — 50 Baud, 02 — 75 Baud, 03 — 110 Baud,
		04 — 135 Baud
2	04	05 — 150, 06 — 300, 07 — 600,
		08 — 1200 Baud
3	08	09 — 1800, 0A — 2400, 0B — 3600,
		0C — 4800 Baud 0D — 7200, 0E — 9600,
		0F — 19,200 Baud
4	10	00 — External Receiver Clock
		10 — Baud-Rate Generator (Normal)
5	20	00 — Data Word Length 8 Bits
6	40	20 — Data Word Length 7 Bits
		40 — Data Word Length 6 Bits
		60 — Data Word Length 5 Bits
7	80	00 — 1 Stop Bit
		80 — 2 Stop Bits (1 if Word Length is 8 bits plus Parity)

```
*  ACIA SUBROUTINES ROUTINES
*
ADATA   EQU    $E180   USE YOUR EQUATES FOR
ASTAT   EQU    $E181   YOUR ACIA REGISTERS
ACMD    EQU    $E182
ACTRL   EQU    $E183
CHAR    EQU    $2000   ONE RAM LOCATION

*  INITIALIZE ACIA

AINIT   LDA    $00    CLEAR STATUS
        STA    ASTAT
        LDA    $0B    ACIA PARAMETERS
        STA    ACMD
        LDA    $16    FOR 300 BAUD, 18 FOR 1200 BAUD ...
        STA    ACTRL
        LDA    ADATA   READ OLD DATA TO START
        RTS

*  TRANSMIT ONE CHARACTER

XMIT    STA    CHAR    SAVE CHARACTER
        LDA    ASTAT   ANY ERRORS ?
        AND    $60    DCD OR DSR ERRORS ?
        BNE    ERRORS  YES, SO SERVICE ERROR
        LDA    ASTAT   NO. READY TO SEND ?
        AND    $10    TRANSMIT READY BIT
        BEQ    NODATA  NOT READY UNLESS BIT SET
        LDA    CHAR    RESTORE CHARACTER
        STA    ADATA   OUTPUT CHARACTER
        CLC            CLEAR CARRY FOR GOOD TRANSMISSION
        RTS

*  RECEIVE ONE CHARACTER

RCVR    LDA    ASTAT   FIRST CHECK FOR ERRORS
        AND    $67    ANY ERROR
        BNE    ERROR   YES
        LDA    ASTAT   DATA READY ?
        AND    $08    RECEIVED DATA BIT
        BEQ    NODATA  BIT SET IF DATA RECEIVED
        LDA    ADATA   DATA AND NO ERROR
        CLC            CARRY CLEAR FOR DATA
        RTS

*  SERVICE ERRORS AND NO DATA/NOT READY

ERROR   PHA            SAVE ERRORS
        LDA    ADATA   READ TO CLEAR ERRORS
        PLA            RESTORE ERRORS INFORMATION
        CMP    $00    SET NOT EQUAL FOR ERROR
NODATA  SEC            SET CARRY FOR NO DATA
        RTS            RETURN WITH ERROR CODE IN A


*
*  SIMPLE PROGRAM TO USE THE SUBROUTINES
*
        JSR    AINIT   INIT ACIA
        .
        .
        LDA    CHR,X   LOAD CHARACTER TO OUTPUT
OUT     JSR    XMIT    OUTPUT VIA ACIA
        BCC    XMOK    TRANSMISSION OKAY IF CARRY CLEAR
        BNE    XERROR  ERROR IF CARRY SET AND NOT EQUAL
        BEQ    OUT     NOT READY IF CARRY SET AND EQUAL
XMOK    .              CONTINUE
        .
        .
IN      JSR    RCVR    INPUT VIA ACIA
        BCC    RCOK    DATA RECEIVED IFCARRY CLEAR
        BNE    XERROR  ERROR IF CARRY SET AND NOT EQUAL
        BEQ    IN      NO DATA IF CARRY SET AND EQUAL
RCOK    .              CONTINUE WITH CHARACTER IN A
        .
XERROR  BRK            BREAK OR ADD ERROR PROCESSING HERE
        .
```

MICRO™

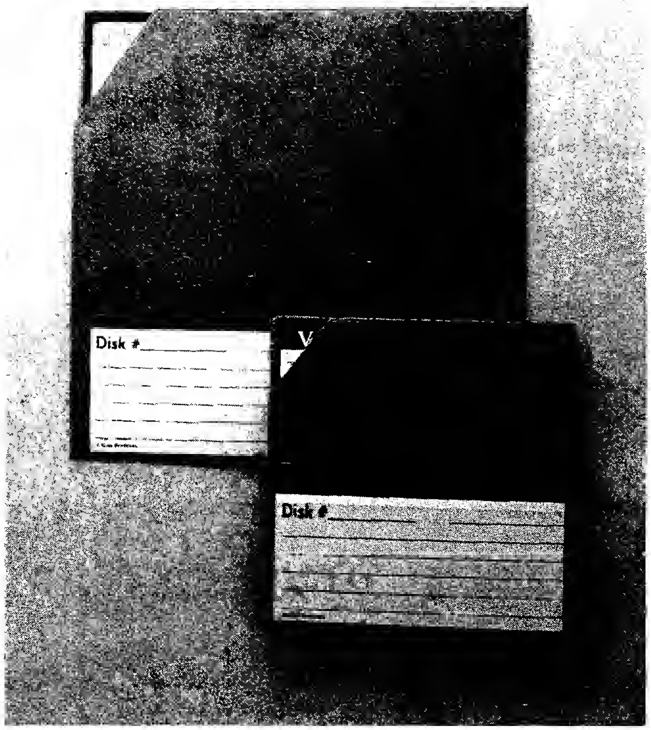
Hardware Catalog

Disk Shield

Info-Guard is a magnetically shielded enclosure that provides protection for disk transportation and storage. It protects flexible disks from distortion, erasure, or degradation of valuable recorded data.

Two models accomodate 8" flexible disks (**$14.50**) and 5¼" flexible disks (**$9.50**).

Available from C-Line Products, Inc., P.O. Box 1278, 1530 E. Birchwood, Des Plaines, IL 60018.



CMOS RAM Module Backup for AIM

Users and designers of 6502-based systems now have available a standard universal memory module to take care of their needs for low cost, low power, novolatile RAM memory. Golden Electronics Inc. has developed a nonvolatile CMOS memory module, **GE65-12K**, to fill a need in Rockwell's AIM/RM65 system of industrial microcomputer cards.

The GE65-12K module provides up to 12K of CMOS RAM with battery backup on a single card. This module has on-card battery backup to provide nonvolatile preservation of data for retention during power loss for such applications as data logging, monitoring, parameter storage, and program backup, according to the company.

The module is fully compatible with both Rockwell AIM and RM65 microcomputer card systems. It is RM65 bus compatible and is provided in a Eurocard version to plug into a Rockwell RM65 card cage or AIM buffer adapter.

Module uses include data preservation in data logging, nonvolatile storage of system parameters and program storage during program development. The GE65-12K, with 2K of memory, is priced at **$235.00**.

For more information contact Golden Electronics Inc., 2133 Yorktown, Ann Arbor, MI 48105. (313) 663-9724.

Kraft Joystick from Kraft Systems Company.

Joystick

The **Kraft joystick** is engineered for high performance and durability, according to the manufacturer. A responsive stick mechanism with spring return provides accurate fingertip control. Cursor positioning is determined by fast-action internal switches. This plug-in unit is designed to reduce hand and finger fatigue. An eight-foot cord is included.

The joystick is compatible with Atari VCS, Atari 400/800, Commodore VIC-20, Commodore 64, and Sears Video Arcade.

Price is **$16.95**. Available from Kraft Systems Company, 450 W. California Ave., Vista, CA 92083; (619) 724-7146.

Low-Cost Graphic Plotter

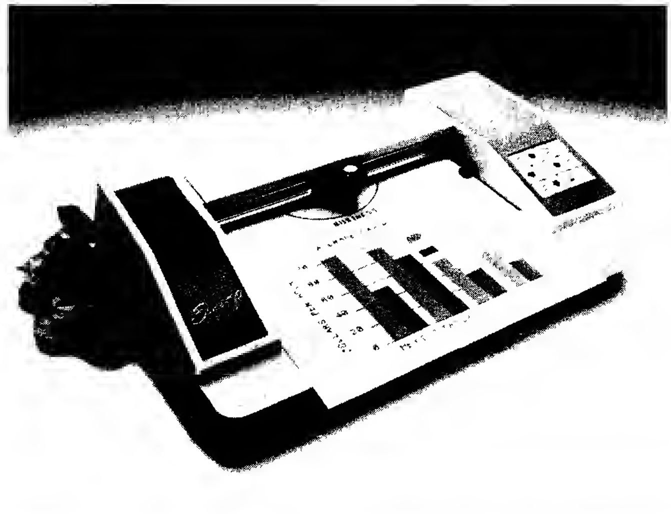
The **Model-100 Sweet-P** personal plotter adds high-resolution, hard-copy graphics capability to Apple, IBM, and other personal or business computers. Interface is via a parallel port similar to the Centronics'. Menu-driven software permits the user to draw, on either paper or overhead transparency, colorful pie charts, bar graphs, line graphs, and technical illustrations.

Software features enable the plotter to define window limits, scale, alphanumeric character size, and character orientation. Provision is made for color fill; digitizing allows the user to scale and draw images traced from photographs, illustrations, or printed material.

The unit has an addressable plotting area of 7.5 x 118 inches, and will accept paper sized from 8½ x 11 inches up to 8½ x 120 inches. Maximum plotting speed is 6 inches per second, and step size is .004'' — or 250 line segments per inch.

The Sweet-P fits into a slimline executive briefcase. The unit is furnished with computer interface, drawing media, and pens, plus a tutorial disk and manual to instruct users in operation and programming.

Price is $795.00. Available from Enter Computer, Inc., 6867 Nancy Ridge Drive, Suite D, San Diego, CA 92121; (619) 450-0601.



Sweet-P Personal Plotter from Enter Computer, Inc.

Apple Disk Drive Add-On

Ghost Drive permits one disk drive to do the work of two—for under $100, according to Aristotle Inc. When the Ghost Drive circuit board is plugged into any of the Apple expansion slots, many software programs will operate on systems with a single drive.

Ghost Drive freezes the processor while the operator switches diskettes in the drive. A lighted toggle switch allows the operator to reactivate the processor. Suggested retail prices is **$79.95.**

Available from Aristotle Industries, Box 21, Norwalk, CT 06853; (203) 853-6683.

RS232 1200 Baud Bell 212 Compatible Modem

SSM TransModem 1200

The **SSM TransModem 1200** features 110/300/1200 baud operation, half and full duplex, auto answer/auto dial, automatic speed detection, touch-tone and pulse dialing, and log-on and password storage. Included with the TransModem is a 6' RJ11 telephone cable that plugs directly into a telephone wall jack and power supply. The TransModem can be used with any computer having an RS232 interface.

For the Apple II Computer SSM offers two Transpak packages. The Transpaks include SSM's TransModem 1200, ASIO serial interface card, all interfacing cables, and TRANSEND 2 or 3. As an added bonus, $220 of special subscription offers to a variety of professional services are included: The Source, America's Information Utility, Dow Jones News Retrieval Service, and Dialog Information Services.

SSM TransModem is priced at **$695.** Transpak 2+ is **$999** and Transpak 3+ is **$1,100.**

Available from SSM Microcomputer Products, Inc., 2190 Paragon Dr., San Jose, CA 95131; (408) 946-7400.

MICRO™

6809 Bibliography

Dr. William R. Dial

129. The Rainbow 2, No. 5 (November, 1982)

Scerbo, Fred, "Now: Get a 3-D View of Space," pg. 8-10.
A TRS-80 Color Computer program to use with 3-D eyeglasses.

Inman, Don, "For More Graphics Info... Turn the Page!," pg. 12-17
A program demonstrating how the 6809-based Color Computer can animate graphics by turning the pages of memory.

Blyn, Steve, "This 'Initial' Experience Can Be Child's Play!," pg. 18.
An educational program for children using the TRS-80 Color Computer.

Delburger, R.T. and D., "Some Beautiful Patterns from 'Down Under'," pg. 23-25.
A moire program for the TRS-80 Color Computer

Hall, Chris, "Now Really Combine Your Graphics and Characters!," pg. 26-32.
A graphics utility for the 6809-based Color Computer.

Speer, Mike, "Now, Let's Hear It For the Cats Too!," pg. 32
A short sound program for the Color Computer.

Wella, Geoff, "If You Went Sailing'...Now Try a Bit of Rowing," pg. 33-36
A graphics program for the 6809-based Color Computer.

130. Systems-68 Magazine, No. 3 (October, 1982)

Wood, Mike, "64 Dynamic RAM Memroy Board with Transparent Refresh," pg 9-16.
Hardware necessary to implement a hardware-refreshed 64K dynamic memory board for use with M6809 or M6800 systems.

Melear, Charles, "Vectoring by Device Using Interrupt Sync Acknowledge with the MC6809/MC6809E," pg. 28-31.
Interrupt techniques for 6809 users.

Pass, E.M., "Some Comments on 6809 Assembler Language Programming for FLEX 9, UNIFLEX, and OS-9," pg. 34-37.
A discussion of certain assembly-language programming concepts related to 6809 systems.

131. Compute! 4, No. 12, Issue 31 (December, 1982)

Anon., "SuperPET Upgrade Board for CBM 8032," pg. 264.
A single board upgrade is available that converts the CBM 8032 into a SuperPET.

132. Microcomputing 6, No. 12 (December, 1982)

Baker, Robert W., "PET-Pourri," pg. 26-28.
A few changes for the Disk Master program when it is run on a 6809-based SuperPET.

133. 80-U.S. Journal 5, No. 12 (December, 1982)

Goodwin, Mark D., "Color Computer Sorting," pg. 63-66.
Sort routine for the 6809-based TRS-80 Color Computer.

134. MICRO, No. 55 (December, 1982)

Steiner, John, "CoCo Bits," pg. 30-31.
Discussion of the TRS-80 Color Computer single disk COPY command.

Malmberg, David and Peterson, Terry M., "SuperPET Characters," pg. 97-98.
Notes on the Waterloo ASCII character set and APL for the 6809-based SuperPET.

135. What Micro?, No. 1 (November/December, 1982)

Staff, "Buyer's Guide to The Micros," pg. 129-159.
Specifications and relative ratings of about 150 different microcomputer systems, including nine systems based on the 6809.

MICRO™

—Next Month in MICRO—

Feature:

Operating Systems

- *Apple Operating Systems*
- *OS-9, A Structured Operating System*
- *Review of the 68000 Educational Board*
- *Comments/Utilities on Color Disk BASIC*

Plus Applications and I/O Enhancements and MICRO's Learning Center!

Advertiser's Index

MICRO INK is not responsible for claims made by its advertisers. Any complaint should be submitted directly to the advertiser. Please also send written notification to MICRO.

National Advertising Representatives

West Coast:
The R.W. Walker Co., Inc.
Gordon Carnie
2716 Ocean Park Boulevard, Suite 1010,
Santa Monica, California 90405 (213) 450-9001

serving: Washington, Oregon, Idaho, Montana, Wyoming, Colorado, New Mexico, Arizona, Utah, Nevada, California, Alaska, and Hawaii (also British Columbia and Alberta, Canada

Mid-West Territory:
Thomas Knorr & Associates
Thomas H. Knorr, Jr.
33 N. Michigan Avenue, Suite 403
Chicago, Illinois 60601 (312) 726-2633

serving: Ohio, Oklahoma, Arkansas, Texas, North Dakota, South Dakota, Nebraska, Kansas, Missouri, Indiana, Illinois, Iowa, Michigan, Wisconsin, and Minnesota.

Middle Atlantic and Southeastern States:
Dick Busch Inc.
Richard V. Busch
6 Douglass Dr., R.D. #4,
Princeton, NJ 08540 (201) 329-2424

Dick Busch, Inc.
Eleanor M. Angone
74 Brookline,
E. Atlantic Beach, NY 11561 (516) 432-1955

serving: New York, Pennsylvania, New Jersey, Delaware, Maryland, West Virginia, Virginia, D.C., North Carolina, South Carolina, Louisianna, Tennessee, Mississippi, Alabama, Georgia, and Florida

MICRO INK is not responsible for claims made by its advertisers. Any complaint should be submitted directly to the advertiser. Please also send written notification to MICRO.


GUARANTEED TOUGH.
LEADING EDGE.
225 Turnpike Street, Canton, Massachusetts 02021
Toll-free: 1-800-343-6833; in Massachusetts (617) 828-8150. Telex 951-624.
Circle No. 83